

A Numerical Study of Fluid Flow Around Two-Dimensional Lifting Surfaces

by

John D. Dannecker

B.S. University of California, Berkeley, 1988

Submitted to the Department of Ocean Engineering
and the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer

and

Master of Science in Mechanical Engineering

at the

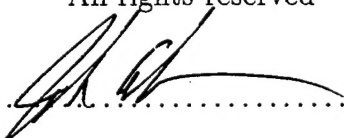
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

©1997 Massachusetts Institute of Technology

All rights reserved

Signature of Author



Department of Ocean Engineering

May 9, 1997

Certified by

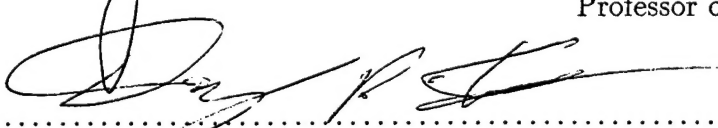


Justin E. Kerwin

Professor of Ocean Engineering

Thesis Supervisor

Certified by

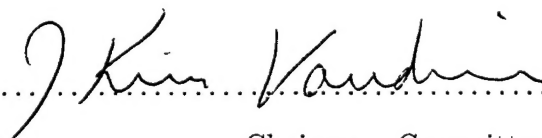


Douglas Hart

Professor of Mechanical Engineering

Thesis Supervisor

Accepted by



J. Kim Vandiver

Chairman, Committee on Graduate Students

Department of Ocean Engineering

Accepted by



Ain Ants Sonin

Chairman, Committee on Graduate Students

Department of Mechanical Engineering

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DTIC QUALITY INSPECTED 1

19970703 068

A Numerical Study of Fluid Flow Around Two-Dimensional Lifting Surfaces

by
John D. Dannecker

Submitted to the Department of Ocean Engineering
and
the Department of Mechanical Engineering
on May 9, 1997, in partial fulfillment of the
requirements for the degrees of
Naval Engineer
and
Master of Science in Mechanical Engineering

Abstract

There are always differences between theoretical and experimental results in the study of lifting surfaces. Bounding box control volume measurements infrequently yield exact conservation of mass or consistent values for lift and drag coefficients. Numerically calculated wakes often differ from experimental data. Quite often, an empirical correction can be applied to fit theory to experiment to account for these differences. However, as the demands for state of the art foil design increase, fluid dynamicists are pressed to look carefully at these inconsistencies in order to improve current design and analysis methods. Using a Reynolds Averaged Navier Stokes (RANS) computer code and a highly refined fluid mesh, one can begin to explore the subtle characteristics of the fluid flow in the entire domain and the details of certain key regions around a foil. Specific areas of great interest are: flow around the trailing edge, flow within the boundary layer, wake profiles and the influence of tunnel wall boundaries in experimental facilities.

The overall goal of this thesis is to resolve some of the discrepancies between theoretical results and experimental data. A computer code has been developed to generate the geometry for the fluid flow domain surrounding an arbitrary foil shape at a specified angle of attack in the MIT Marine Hydrodynamics Laboratory (MHL) water tunnel. This geometry is provided as input data for the RANS solver. A suite of software tools are developed to provide post processing analysis to compare the RANS solution with other numerical techniques and experimental measurements.

Through the use of case studies, the numerical results of the RANS code are compared with recent MHL experimental data and other computational tools. A comparison is made between the experimental and RANS code results using a control volume analysis. Boundary layer and wake profiles are also compared. A correction scheme is developed to extrapolate experimental measurements to unbounded fluid flow.

Thesis Supervisor: Justin E. Kerwin
Title: Professor of Naval Architecture

Thesis Supervisor: Douglas Hart
Title: Professor of Mechanical Engineering

Acknowledgments

This thesis could not have been completed without the encouragement and support of my wife Michelle and my two children, David and Marina. I am eternally grateful for the patience and endurance during my studies here at MIT.

I thank Professor Jake Kerwin for his gentle guidance and enthusiasm.

To Scott Black, Todd Taylor, Gerard McHugh, Rich Kimball, and the many other Propnuts who pointed me in the right direction now and then.

To Bill Milewski for pointing out to me that ninety percent of computational fluid dynamics is getting the geometry right.

To Mr. Gordon Stevens, my high school physics teacher, who first taught me the importance of dimensional analysis and that science can be a lot of fun.

Lastly, I am grateful to the United States Navy for providing the opportunity and funding for me to pursue higher education.

Contents

1	Introduction	16
1.1	Overview	16
1.2	Background	17
1.2.1	Why Is There Interest in 2-D Flow?	17
1.2.2	A Recent Design Problem	17
1.3	Motivation for Modeling 2-D Foils With RANS	18
1.4	Objectives	19
1.4.1	Rapid Generation of Flow Geometry	20
1.4.2	Resolving Differences Between Experiments and Numerics	20
1.5	Organization	21
2	Theory	22
2.1	Overview	22
2.2	Two-Dimensional Lifting Flows	22
2.3	Linearized Two-Dimensional Theory	24
2.3.1	Lift	25
2.3.2	Drag	26
2.3.3	Lift and Drag by Pressure Integration	26
2.3.4	Viscous Effects on Lift	27
2.4	Vortex Lattice Methods(VLM)	28
2.4.1	Cosine Spacing	28

2.4.2	Obtaining the Vortex Lattice Solution	29
2.4.3	Method of Images Applied to VLM	31
2.5	Momentum Theory for Lift and Drag Calculations	33
2.5.1	Bounding Box Analysis	33
2.5.2	Drag by Wake Defect	34
2.6	Reynolds Averaged Navier Stokes Equations	36
2.6.1	Derivation of the RANS Equations	36
2.6.2	Turbulence Modeling	37
2.7	Boundary Layer Flows	39
2.7.1	Characterization of Boundary Layers	39
2.7.2	Laminar Boundary Layer Flows	40
2.7.3	Turbulent Boundary Layers	42
2.7.4	The y^+ Parameter	43
2.7.5	The Law of the Wall	43
3	Numerical Methods Development	46
3.1	Overview	46
3.2	Overview of Grid Generation	47
3.2.1	Fundamental Concepts	47
3.2.2	Geometric Limitations of DTNS2D	49
3.3	Development of the Computer Code FIT2D	50
3.3.1	Functional Requirements	50
3.3.2	Structure of FIT2D	52
3.3.3	Selecting Required Input	52
3.3.4	Splining and Element Distribution Methods	54
3.3.5	Defining Domain Boundaries	55
3.3.6	Isoparametric Interpolation	56
3.4	FIT2D Output	57

3.4.1	Tecplot Files	57
3.4.2	INMESH Input Files	58
3.5	Adjustment of Zone Boundaries in the Wake	59
3.5.1	Implementation of the ADAPT Subroutine	59
3.5.2	Adding Image Vortices	60
3.5.3	Convergence of the Vortex Lattice Method	61
3.5.4	Adapting Wake Zone Boundaries Using RANS Output	63
3.6	Using a Poisson Solver to Refine Grid	63
3.6.1	The Program INMESH	63
3.6.2	Correction of INMESH output for use in DTNS2D	65
3.7	The RANS Solver	66
3.7.1	Overview of DTNS2D	66
4	Data Post Processing Methods	67
4.1	Introduction	67
4.2	UNNS2D	67
4.3	Computing Lift and Drag	68
4.4	Bounding Box Calculations	68
4.5	Extracting Velocity Profiles in the Boundary Layer	69
4.5.1	Post Calculation of y^+	69
4.6	Extracting Velocity Profiles in the Wake	70
5	Results	71
5.1	Validity of the Results	72
5.1.1	Sources of Error	72
5.1.2	General Comments Regarding Comparison with Tunnel Experiments	72
5.2	Case Study I: The HRA Foil	73

5.2.1	Validation Check of Grid Adequacy	74
5.2.2	Unbounded Flow Comparison	76
5.2.3	Bounded Flow Comparison	76
5.2.4	Bounding Box Comparison	77
5.2.5	Wake Profile Comparison	78
5.2.6	Relating Bounded Measurements to Unbounded Characteristics	79
5.3	Case Study II: Foil With A Cupped Trailing Edge	84
5.3.1	Validation Check of Grid Adequacy	85
5.3.2	Unbounded Flow	87
5.3.3	Bounded Flow	89
5.3.4	Tunnel Wall Boundary Layer Comparison	92
5.3.5	Separated Flow	93
5.3.6	Developing An Experimental Test Plan	94
6	Conclusions and Recommendations	98
6.1	Conclusions	98
6.2	Recommendations	99
A	FIT2D Program Listing	101
B	Sample FIT2D Input Files	150
B.1	Bounded Foil (<i>fname.ctrl</i>)	150
B.2	Unbounded Foil (<i>fname.ctrl</i>)	151
B.3	Sample Foil Geometry File (<i>fname.foil</i>)	152
C	Marine Hydrodynamics Lab Water Tunnel Geometry	153
C.1	System Overview	153
D	GETWAKE Program Listing	155

E	PATCH Program Listing	158
F	Case Study Foil Offsets	160
F.1	Case Study I: The HRA Foil	161
F.2	B-1 Foil With Cup Modification To Trailing Edge	162

List of Figures

2-1	2-D Foil Coordinate System	23
2-2	Point Vortex in Uniform Stream	25
2-3	Viscous Boundary Layer on 2-D Foil	27
2-4	Vortex Lattice on a 2-D Mean Line	29
2-5	Vortex Lattice Near Walls With Images	32
2-6	Rectangular Contour Around 2-D Hydrofoil	34
2-7	Spalding Formula for <i>Law of the Wall</i>	44
3-1	Sample of the discretized flow domain of a foil in a water tunnel. (Only 1/3 of the total grid lines are reproduced in the lower frame)	48
3-2	Sample geometry of a DTNS2D zone. (Only 1/3 of the total grid lines are reproduced)	50
3-3	Typical DTNS2D zonal boundaries for a foil in a tunnel	51
3-4	Isoparametric Mapping Schematic of Interior Zonal Points	57
3-5	Display of <i>rotate.plt</i> data	58
3-6	Vortex Lattice Geometry with Extracted Wake Line	60
3-7	Comparison of wakes for VLM(unbounded) to VLM with Images(bounded)	61
3-8	Vortex Lattice Panel Convergence Curve	62
3-9	Vortex Lattice Image Pair Convergence Curve	62
3-10	Sample Comparison of VLM Wake and RANS Wake Zone Adaption	64
3-11	Typical Grid Spacing Before and After Smoothing with INMESH	65

4-1	Pressure Distribution Dependence on y^+ Parameter	70
5-1	The HRA Foil Shape	74
5-2	Comparison of DTNS2D Computed B-L Profile with Spalding Formula for HRA Foil	75
5-3	Comparison of Pressure Distribution on HRA Foil Using DTNS2D and PAN2D-BL($Re = 3 \times 10^6, AOA = 1^\circ$)	77
5-4	Comparison of Normal Velocity Component Around Contour Box for DTNS2D and HRA Experimental Results($Re = 3 \times 10^6, AOA = -0.636^\circ$)	79
5-5	Comparison of Wake Profiles for DTNS2D and HRA Experimental Results($Re = 3 \times 10^6, AOA = -0.28^\circ$)	80
5-6	Summary of Results for Lift Coefficient vs. AOA for the HRA Foil . .	81
5-7	Summary of Results for Drag Coefficient vs. AOA for the HRA Foil .	82
5-8	Lift Curve With Correction Factors Applied for HRA Foil	83
5-9	The B-1 Foil With Cupped Trailing Edge Modification	85
5-10	Comparison of DTNS2D Computed B-L Profile with Spalding Formula for B1 Cupped Foil	86
5-11	Velocity Contours Near Trailing Edge of Cupped Foil(unbounded flow)	87
5-12	Pressure Contours Near Leading Edge of Cupped Foil(unbounded flow)	88
5-13	Comparison of Pressure Distribution on Cupped Foil Obtained by Two Different RANS Solvers and PAN2D-BL	89
5-14	Velocity Contours Near Trailing Edge of Cupped Foil(bounded flow) .	90
5-15	Pressure Contours Near Leading Edge of Cupped Foil(bounded flow)	91
5-16	Comparison of Pressure Distribution on Cupped Foil Obtained by Three Different RANS Solvers	92
5-17	Comparison of Pressure Distribution on Cupped Foil for Bounded and Unbounded Flow	93
5-18	u Contours for Cupped Foil in Bounded Flow	94

5-19 Upper Tunnel Wall Boundary-Layer Profiles	95
5-20 Upper Tunnel Wall Boundary-Layer Displacement Thickness and Mo- mentum Thickness	96
5-21 Streamlines Near Trailing Edge of Cupped Foil(DTNS2D Solution) .	97
5-22 Streamlines Near Trailing Edge of the B1 Cupped Foil(C.I. Yang So- lution)	97
C-1 MIT MHL Water Tunnel	153
C-2 MIT MHL Water Tunnel Test Section	154

List of Tables

3.1	Batch Input Variables for FIT2D	53
5.1	Post Processing Check of y^+ for HRA Foil Grid	75
5.2	CONTOUR and FLD2D Results for Lift and Drag Characteristics of the HRA Foil	78
5.3	Lift Slope and Intercept Data for HRA Foil Calculations	81
5.4	Post Processing Check of y^+ for B-1 Cupped Foil Grid	86
5.5	Comparison of Unbounded C_L & C_D Calculations for B1 Cupped Foil ($Re = 3 \times 10^6, AOA = 0.5^\circ$)	89
5.6	Comparison of Bounded C_L & C_D Calculations for B1 Cupped Foil ($Re = 3 \times 10^6, AOA = 0.5^\circ$)	91

Nomenclature

Mathematic Notation

c	Chord Length
C	Closed Contour
C_d	Total Drag Coefficient
C_{dp}	Drag Coefficient due to pressure forces
C_{dv}	Drag Coefficient due to viscous forces
C_l	Lift Coefficient
dv	jump velocity across vortex sheet
F_D	total drag on lifting surface
F_L	lift force on lifting surface
F_{Dv}	viscous drag on lifting surface
F_{Dp}	pressure drag on lifting surface
\mathbf{i}	normal vector in x -direction
\mathbf{j}	normal vector in y -direction
\mathbf{n}	normal unit vector on a surface
\mathbf{n}_x	x -comp. of normal vector on a surface
\mathbf{n}_y	y -comp. of normal vector on a surface
P	pressure
$\Re \quad \frac{ul}{\nu}$	Reynolds number
s	curvilinear distance along camber surface
S_{len}	total arc length

Mathematic Notation(continued)

S	domain of foil surface
\mathbf{t}	tangent unit vector
\mathbf{t}_x	x -comp. of tangent vector on a surface
\mathbf{t}_y	y -comp. of tangent vector on a surface
τ	fluid shear stress
U_{inf}	Freestream velocity
u	local x -velocity
u_τ	$\sqrt{\tau_{ns}/\rho}$ Friction Velocity
v	local y -velocity
\mathbf{V}	velocity
Γ	dimensional circulation
γ	vorticity
γ_b	bound vorticity
γ_f	free vorticity
μ	dynamic viscosity
ν	kinematic viscosity
ρ	density of fluid
∇^2	Laplace Operator

Abbreviations

2-D	Two Dimensional (flow)
AOA	Angle of Attack
ATD	Advanced Technology Demonstration
B-L	Boundary Layer
CPU	Central Processing Unit
DDG	Destroyer-Guided Missile
DTNS	David Taylor Navier Stokes
DTNS2D	David Taylor Navier Stokes Solver 2-Dimensional
HRA	Hydrodynamic Research Associates
$k - \varepsilon$	Two equation energy and dissipation turbulence model
LDV	Laser Doppler Velocimetry
MHL	Marine Hydrodynamics Laboratory
MIT	Massachusetts Institute of Technology
N-S	Navier-Stokes (equations)
RAM	Random Access Memory
RANS	Reynolds-averaged Navier-Stokes (flow solver)
SOR	Successive-Over-Relaxation (solution method)
TVD	Total-Variation-Diminishing
VLM	Vortex Lattice Method

Chapter 1

Introduction

1.1 Overview

Recent application of advanced computational methods to design new hydrofoil¹ sections for propeller blades has yielded inconsistent results. There is a need in the hydrodynamic research community to undertake an in depth study of two-dimensional lifting surfaces since these sections are the fundamental building blocks of most three dimensional design methods. Additionally, a sound methodology should be applied to make effective comparisons between numerical and experimental results. Once the differences between numerical predictions, experimental measurements and real application are understood, those lessons can be incorporated into computational methods. Some improvement will come from better understanding of fluid flow within boundary layers near the foil surface. Other areas in need of refinement are the effects of wake diffusion and flow separation near the trailing edge.

Ultimately, yielding these improvements will be a multi-step process that involves significant computational and experimental effort far beyond the scope of a single thesis. The work herein represents the first few steps towards the goal of developing a more complete understanding of the subtleties of fluid flow around two-dimensional foils.

¹Throughout this thesis, the words hydrofoil, 2-D lifting surface and foil will be used as synonymous terms.

1.2 Background

1.2.1 Why Is There Interest in 2-D Flow?

Presently, the hydrodynamic research community has turned its attention back to the study of two-dimensional lifting surfaces. The renewed interest is derived from recent attempts to design advanced hydrofoil sections for propeller applications that demonstrate improved cavitation characteristics without incurring a significant drag penalty.

Recent work in the MIT MHL[17] water tunnel by Jorde[16] and Kimball[19] on foils with non-traditional camber distributions and unique trailing edges indicate that the above goals may be attainable. However, they also point out that our knowledge of two-dimensional flow around foils is incomplete. Bloch[7] developed a methodology for adding corrections to inviscid propeller design codes to account for anti-singing trailing edges. Several issues have come to light. First, the current computer design codes work well for conventional foils. However, the current codes do not work well for foils with cupped or blunt trailing edges or for foils with advanced camber distributions designed to delay cavitation inception.

1.2.2 A Recent Design Problem

As part of an advanced technology demonstration(ATD), new high speed propellers were designed and tested at the Carderock Division, Naval Surface Warfare Center. The design effort required to achieve the improved propeller performance using advanced blade sections was monumental. The design process involved iterating several times between using propeller design computer codes and model testing[3]. The costs and time associated with this type of design procedure are prohibitive.

It is not practical to expect that any commercial operator buying a propeller for a ship could fund such a research and development effort. But, the commercial operators do want the advantages these new foil sections offer. The way to make

designing with advanced foil sections cost effective is to eliminate the costly iteration between design computer codes and model testing by developing a database that generalizes these new geometries as suggested by Bloch. Prudent application of model testing and RANS analysis for a broad geometry of foil shapes could provide the necessary data to formulate correction routines for standard propeller design codes without significantly increasing the computation time for a converged solution.

1.3 Motivation for Modeling 2-D Foils With RANS

An incompressible Reynolds Averaged Navier Stokes (RANS) solver was chosen for the bulk of the numerical analysis in this thesis. A RANS computer code solves the equations of motion for each fluid element throughout an entire domain. When compared to other solution methods, such as an inviscid panel method, RANS is very time consuming and requires large amounts of random access memory (RAM) and central processing unit (CPU) time. This is a significant disadvantage, especially when one wants to analyze several foils at several angles of attack and Reynolds numbers. But, there are many aspects of a RANS solution that are attractive.

The voluminous output from a RANS computer code contains all of the flow characteristics for every fluid element throughout the entire domain such as: Velocity, Pressure, and Shear Stress. Provided the flow domain is discretized with sufficient resolution, you can also extract and measure subtleties of the flow like boundary layer velocity profiles and thickness, and flow separation under adverse pressure gradients. It is desirable to be able to study the characteristics of a viscous wake behind a hydrofoil. Also, with the abundance of data available in the solution, it is easy to make a direct comparison between conditions calculated at a prescribed location in the flow field and those measured in a geometrically similar experiment.

A large part of this study involves characterizing the differences between numerical solutions and experimental measurements. In an unbounded flow, such as a 2-D

foil fixed in a uniform stream, RANS codes provide a good validation for computer programs such as PAN2D [21] and XFOIL [10] which are both inviscid panel methods coupled with integral boundary layer solvers. Neither PAN2D nor XFOIL is currently capable of modeling a foil in a flow constrained by walls including the viscous effects of the boundary layers, which is the case in water tunnel experiments. A RANS code can serve as a *liaison* between unbounded numerical codes and the bounded case of experiments. For an equivalent foil geometry, the RANS code can be used to characterize the differences in lift, drag and other properties for both bounded and unbounded flows. A methodology for doing this is presented in Chapter 5.

1.4 Objectives

As was alluded to in Section 1.3, one overall goal in this thesis is to provide a scheme for feeding back results and measurements taken in experiments as corrections that can be applied to the computationally efficient inviscid panel methods. Several steps need to occur before this goal can be realized in a substantial way. Methods need to be developed to efficiently generate input files for the RANS solver. As will be shown in Chapter 3, accurate geometric representation of a flow domain is perhaps the most demanding part of obtaining the solution. Accordingly, a great deal of effort was devoted to ensuring that the RANS domain geometry was exactly the same as the inviscid computer codes and the experimental setups. Computer codes need to be developed to reduce the output data to a tractable and meaningful form. The following are the critical path or the enabling objectives used to achieve the above goals:

1. Review recent MIT MHL Water Tunnel experimental results for advanced section two-dimensional hydrofoils.
2. Write a computer code that generates the fluid flow geometry input files for the RANS solver.

3. Analyze foils in unbounded flows using the RANS code and PAN2D.
4. Analyze foils in bounded flows using the equivalent geometry of the MIT MHL.
5. Develop corrective factors to apply to the inviscid solvers which are based on the differences found between the RANS and experimental results.
6. Demonstrate how RANS solutions can be used to formulate experimental test plans.

1.4.1 Rapid Generation of Flow Geometry

To support ongoing research in the MIT Marine Hydrodynamics Laboratory, a two-dimensional lifting surface analysis tool is developed in this thesis to accurately model the fluid flow around hydrofoils in the water tunnel. It is very useful to have good predictions of a hydrofoil's performance characteristics prior to conducting an experiment. Thorough empirical evaluation of hydrofoils requires taking many measurements with varying geometry and Reynolds numbers. It is common to take measurements for a single foil geometry at several angles of attack and Reynolds number. Therefore, one requirement for any computational tool used in the MHL is that it be easy to change the foil angle of attack and other test conditions such as scale and Reynolds number. Part of the work in this thesis is the development of the computer code FIT2D(Foil In Tunnel, Two-Dimensional). FIT2D is an interactive fluid mesh geometry generator. The user can arbitrarily specify: angle of attack, grid resolution, and Reynolds number as well as many other lesser parameters. The output files generated are used as input for a Poisson equation ($\nabla^2\phi = Const$) grid refinement program. After the grid is refined, it becomes input for a RANS solver for analysis.

1.4.2 Resolving Differences Between Experiments and Numerics

In sections 1.3 and 1.2.1, it is asserted that to make the current foil design computer codes work better, corrections should be incorporated to account for the physical effects that are not modeled in computationally efficient inviscid solutions. Kimball

found that there is uneven boundary layer growth on tunnel walls due to the presence of a foil that is generating lift[19]. This difference between the upper and lower walls affects the flow and the resulting lift and drag measurements. So there are really two sides to the correction scheme. One is using RANS to quantify the effects that the tunnel walls have on the foil lift and drag measurements as a result of uneven boundary layer growth and the potential flow imaging effect. The other is using the experimental measurements to apply corrections to computer codes for physics that are not captured by the numerics such as re-attachment of separated flows, transition back to a laminar boundary layer, and vortex shedding phenomena.

1.5 Organization

In Chapter Two, all of the relevant theory of two-dimensional lifting surfaces is outlined. It provides the mathematical statement of the lifting problem. Similarly, Chapter Three is a detailed description of the numerical aspects of the computational solution.

Chapter Four outlines the methodology for post processing all of the RANS data. It provides an overview of the suite of programs that were developed and used to calculate lift and drag, and analyze wake profiles and boundary layers.

Chapter Five compares and contrasts the RANS output to other numerical methods and experimental measurements. Results of case studies for two different foils are presented. Figures, Graphs, and Tables are used to demonstrate the differences and similarities that occur. A methodology is proposed for applying corrections hydrofoil computer programs to gain better agreement with precise experimental results and real applications. A demonstration of how RANS solutions can be used to formulate experimental test plans is presented.

Chapter Six is a summary of results and conclusions. Future topics and directions for research in this area are discussed.

Chapter 2

Theory

2.1 Overview

This study focuses on 2-D fluid flow, therefore it is appropriate that this chapter outlines and develops the pertinent fluid mechanics theories for lifting surfaces and boundary layers. These theories have been previously derived in many texts and technical papers. The developments and derivations contained in this thesis represent the specific application to the problem of thin non-cavitating hydrofoils in bounded and unbounded incompressible two-dimensional flow domains.

First, the basic lifting problem is defined as a boundary value problem and the geometry is specified. Subsequent sections draw attention to aspects of lifting surface theory that are fundamental to the computational methods employed and to the physical phenomena that are modeled.

2.2 Two-Dimensional Lifting Flows

Lifting surfaces have many applications in marine hydrodynamics. Two dimensional foil sections are used extensively in the design of rudders, keels, skegs, propellers, and other appendages, such as ducts, where a desired force is generated normal to the onset flow.

This study is limited to a “typical” lifting surface as described in Newman [24] where the thickness is much smaller than the chord length ($t(x) \ll C$). Further, only

incompressible two-dimensional flows are considered where the span of the foil is of infinite length and the resultant flow over the surface is chord-wise in direction. Figure 2-1 shows how the foil geometry is defined with respect to the coordinate axes and the onset fluid flow. The foil is fixed in a stationary reference frame. The velocity

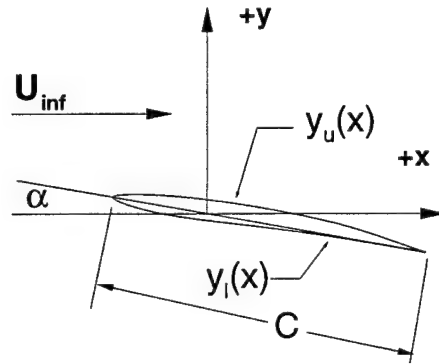


Figure 2-1: 2-D Foil Coordinate System

field (U_{inf}) propagates in the direction of the positive x -axis, and accordingly the leading edge of the foil points opposite to the onset flow. Although the pivot point is arbitrary for an unbounded flow, it becomes important when walls are present close to the foil surface. In this work, care was taken to specify the exact x, y location of the pivot point for comparing numerical results to those obtained in the MHL water tunnel. The angle of attack (α) is defined as the angle that an imaginary line drawn from the leading edge to the trailing edge makes with the x -axis. Here it is assumed that the “nose to tail” line intersects the fore and aft end of the mean camber line at the zero angle of attack. If the trailing edge is blunt, beveled or has some other treatment such as a splitter plate, the mean camber line ends at the midpoint of a line drawn from the upper and lower points of the trailing edge surface. The mean camber line is defined as:

$$y_m(x) = \frac{1}{2}(y_u(x) + y_l(x)) \quad (2.1)$$

Foil thickness is defined as:

$$y_t(x) = y_u(x) - y_l(x) \quad (2.2)$$

Steady state flow is assumed for all cases. The Kutta Condition[18] applies at the trailing edge requiring that the fluid velocity be finite. With the above limitations and the stated geometry, the following boundary value problem results:

$$\nabla^2 \phi = 0 \quad (2.3)$$

$$\mathbf{V} = 0, \text{ on } y_u(x) \text{ and } y_l(x) \quad (2.4)$$

$$\nabla \phi < \infty, \text{ at the trailing edge, Kutta Condition} \quad (2.5)$$

$$\mathbf{V} = 0, \text{ on walls if present} \quad (2.6)$$

$$\nabla \phi = U_{inf} \mathbf{i}, \text{ as } x, y \rightarrow \infty \quad (2.7)$$

Equations 2.3 - 2.7 are the complete mathematical statement of the lifting problem for a stationary foil with onset flow. Two types of solutions to this problem are pursued. A Reynolds Averaged Navier Stokes solver is used to solve the steady state viscous flow around the foil. The other method is a potential flow vortex lattice method where the no-slip rigid boundary conditions of Equations 2.4 and 2.6 are relaxed. In inviscid potential flow, the wall and foil surface boundary conditions become:

$$\mathbf{V} \cdot \mathbf{n} = 0 \quad (2.8)$$

Now that the boundary value problem is defined for the viscous and inviscid problems, the solutions can be formulated.

2.3 Linearized Two-Dimensional Theory

The use of linear potential theory is very powerful for analyzing lifting flows. A simple example of a lifting potential flow is a point vortex fixed in a uniform stream as shown in Figure 2-2. The linear potential for a free stream with a point vortex located at

x_o, y_o is:

$$\Phi = U_{inf}x + Im \frac{\gamma}{2\pi} (\log(x - x_o + i(y - y_o))) \quad (2.9)$$

This potential is a solution to the Laplace equation (2.3) throughout the flow field except at the location of the point vortex. Using the principle of superposition, a more complicated flow can be formulated by adding a group of simple flows together that satisfy the kinematic boundary conditions and Laplace.

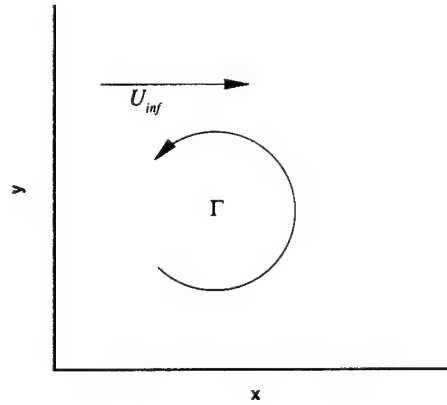


Figure 2-2: Point Vortex in Uniform Stream

2.3.1 Lift

In potential flow the lifting force acting on a foil is obtained using the *Kutta-Joukowski* theorem, which states that for any two-dimensional body, moving with constant velocity in an unbounded inviscid fluid, the hydrodynamic pressure force is directed normal to the velocity vector and is equal to the product of the fluid density, body velocity and the circulation about the body[24]. The mathematical statement of this theorem applied to thin foils is:

$$L = \oint (p - p_{\infty})dx = \rho U \oint u \, dx = \rho U \Gamma \quad (2.10)$$

In a potential flow that is formulated by a distribution of discrete point vortices, application of 2.10 is a simple matter of summing up the strengths of the point

vortices enclosed by a simple contour.

2.3.2 Drag

In a pure inviscid potential flow solution, there is no pressure or *form* drag. However, when fluid viscosity is added to the solution, a viscous boundary layer (section 2.7) is formed near the foil surface. Drag on the foil is present as a result of two phenomena. First, shear stresses in the boundary layer cause skin friction drag. Second, due to the presence of the boundary layer, the pressure recovery at the trailing edge is incomplete which causes a drag force. The pressure drag is hard to calculate or measure because the pressure changes that cause the drag are isolated to a small area near the trailing edge of the foil. Also when compared to the lift, drag is usually a very small quantity. The errors associated with computing the drag are nearly equal to the drag itself.

Two other methods of computing the drag are presented later in sections 2.5.1 and 2.5.2. These methods are not as accurate as direct integration of pressure and shear stress but provide additional verification of the integrated quantities.

2.3.3 Lift and Drag by Pressure Integration

When the viscous effects are included, *Kutta-Joukowski* does not apply for calculating lift. The drag is no longer zero. However, if the pressure and shear stress in the fluid adjacent to the surface (S) are known, the lift and drag can be obtained by direct pressure integration on the foil surface around the perimeter of the foil. The lift and drag forces on the foil due to pressure integration are:

$$\mathbf{F}_{Dp} = \int_S P \cdot \mathbf{n}_x dS \quad (2.11)$$

and

$$\mathbf{F}_L = \int_S P \cdot \mathbf{n}_y dS \quad (2.12)$$

where S is the foil surface and P is the fluid pressure at the foil surface. In addition to the normal pressure to the surface, there is a viscous shearing stress (τ_{ns}) acting

tangent to the surface. In a similar fashion to the above equations, shearing forces can be calculated by integrating around the foil and resolving into x and y components. Typically, the y component of the shearing forces is neglected because the integral quantity of these forces is insignificant compared to the y component of the pressure forces. The drag force on a foil due to viscous shearing forces is:

$$\mathbf{F}_{Dv} = \int_S \tau_{ns} \cdot \mathbf{t}_x dS \quad (2.13)$$

The total drag on the foil is the sum of the pressure and viscous drag terms, or:

$$\mathbf{F}_D = \mathbf{F}_{Dp} + \mathbf{F}_{Dv} \quad (2.14)$$

2.3.4 Viscous Effects on Lift

A detailed overview of the viscous effects on lift is contained in Coney[9]. Some pertinent aspects of his discussion are presented here to provide needed clarity. As shown in Figure 2-3¹, a foil in the presence of a real fluid flow will have a thin viscous boundary layer formed on its surface. This, in essence, alters the effective thickness

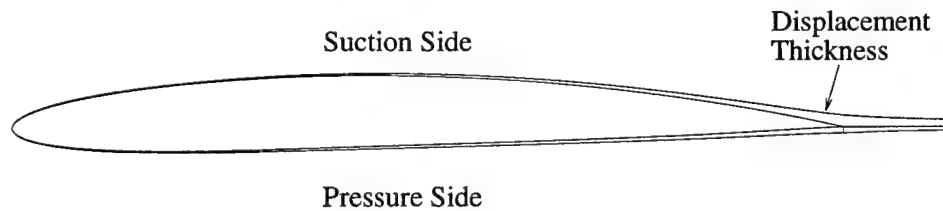


Figure 2-3: Viscous Boundary Layer on 2-D Foil

and camber distribution of the foil. The camber is reduced on the aft portion of the foil. In general, for the same foil geometry, the difference between the potential flow and viscous flow solution is that the change in camber distribution will cause an apparent reduction in lift coefficient. Boundary layer characteristics are covered in more detail in section 2.7

¹Figure 2-3 created by Scott D. Black

2.4 Vortex Lattice Methods(VLM)

The basic concepts of vortex lattice theory are contained in Newman [24] and Kerwin [18]. In light of the long computation time to obtain a RANS solution, it is a prudent step to incorporate a vortex lattice analysis code into the computer code that generates the geometry for the RANS solver. This serves two purposes. First, it provides an initial estimate for the lift coefficient(C_l) and the circulation distribution(γ) on a foil. Second, it provides the dividing streamline in the wake by extraction of the field point velocity due to the circulation and free stream potential. This is desirable because the numerics of the RANS solver works better if the fluid domain discretization is adapted to the location and trajectory of the wake.(See Section 3.7)

Vortex lattice methods are simple and efficient. The earliest vortex lattice method is attributed to Falkner[12] as referenced by Kerwin[18]. The derivation presented here is analogous to Kerwin's method. The primary difference is that the actual mean line of the foil is used vice the linearized foil surface.

The formulation of the vortex lattice method is basic. The mean camber line of a given foil is determined using equation 2.1. The camber line is broken up into a discrete number of elements of *panels*. The panels are spaced using a "cosine" method where the panels at the leading and trailing edge of the foil are smaller than the middle panels. A point vortex of unknown strength Γ_m is located at the mid-point of each of the m panels. Control points where the kinematic boundary conditions are imposed are located at the end of each panel. Figure 2-4 is an example of a vortex lattice distributed on a mean camber line.

2.4.1 Cosine Spacing

Cosine spacing is commonly used in many lifting surface and vortex lattice methods as the spacing algorithm for distributing point vortices and control points. It is a

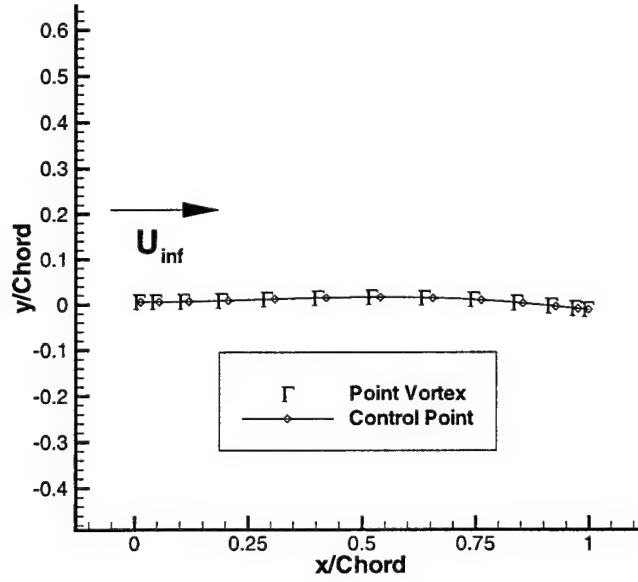


Figure 2-4: Vortex Lattice on a 2-D Mean Line

simple method where an auxiliary variable \tilde{s} is defined such that:

$$s = \frac{S_{len}}{2}(1 - \cos \tilde{s}) \quad (2.15)$$

where $\tilde{s} = 0$ is the leading edge of the mean line and $\tilde{s} = \pi$ is the trailing edge. In the auxiliary coordinate system, the interval of the arc is divided into N equal panels each of $\tilde{s} = \pi/N$ length. To establish the locations for the point vortices and the control points the following equations are used:

$$s_v(n) = \frac{S_{len}}{2} \left[1 - \cos\left(\frac{(n - 1/2)\pi}{N}\right) \right] \quad (2.16)$$

$$s_c(n) = \frac{S_{len}}{2} \left[1 - \cos\left(\frac{n\pi}{N}\right) \right] \quad (2.17)$$

2.4.2 Obtaining the Vortex Lattice Solution

The boundary condition on the foil is prescribed by equation 2.8. The strength of the point vortices are to be determined such that this boundary condition is satisfied at each of the control points. First it is instructive to have the expression for the

velocity field induced at a point x, y by a point vortex located at the origin[24].

$$\mathbf{V} = -\frac{\Gamma}{2\pi} \frac{\mathbf{i}y - \mathbf{j}x}{(x^2 + y^2)} \quad (2.18)$$

This equation can easily be converted to the velocity field induced by a point vortex located at x_v, y_v on a control point located at x_c, y_c by making a simple substitution of the difference in the coordinates for x and y .

$$\mathbf{V}_{\text{at } c \text{ due to } v} = -\frac{\Gamma_v}{2\pi} \frac{\mathbf{i}(y_c - y_v) - \mathbf{j}(x_c - x_v)}{[(x_c - x_v)^2 + (y_c - y_v)^2]} \quad (2.19)$$

To extend the expression to apply to the vortices and control points distributed as per equations 2.16 and 2.17, simply insert the appropriate indices. Then the solution of the vortex lattice can be formulated into a mathematical statement. The velocity induced at the n^{th} control point by all the N vortices dotted with the normal vector is equal to *minus* the free stream velocity dotted with the normal vector.

$$\frac{-1}{2\pi} \sum_{m=1}^N \Gamma_m \frac{\mathbf{i}(y_c(n) - y_v(m)) - \mathbf{j}(x_c(n) - x_v(m))}{[(x_c(n) - x_v(m))^2 + (y_c(n) - y_v(m))^2]} \cdot \mathbf{n}_n = -U_{\text{inf}} \mathbf{i} \cdot \mathbf{n}_n \quad (2.20)$$

Equation 2.20 can be written for each of the control points. This represents a set of N simultaneous equations which can be solved for the unknown vortex strengths (Γ_n). Separating out the Γ_n terms as a separate component, the terms that are left formulate a square matrix that represents the induced velocity caused by each point vortex on every control point. This matrix is called the influence coefficient matrix (\mathbf{A}). Equation 2.20 can be cast in matrix form as:

$$\mathbf{A} \cdot \mathbf{\Gamma} = \mathbf{U} \quad (2.21)$$

where $\mathbf{\Gamma}$ is the array of point vortex strengths and \mathbf{U} is the array of boundary conditions at the control points. The \mathbf{A} matrix can be reduced using Gaussian elimination or any convenient algorithm for matrix inversion[4]. Once the matrix is inverted, the vortex strengths can be determined.

$$\mathbf{\Gamma} = \mathbf{A}^{-1} \cdot \mathbf{U} \quad (2.22)$$

This completes the treatment of the 2-D vortex lattice method for a foil in an unbounded fluid. The next aspect to consider is the development of wall corrections for when a vortex lattice is located near a wall.

2.4.3 Method of Images Applied to VLM

Many of the cases analyzed involve 2-D foils which are in close proximity to walls. This is the case for testing a foil in a water tunnel. Therefore, the unbounded vortex lattice method needs to be modified to account for the presence of the walls. There are two options for modeling the walls. The first is to put vortex lattice panels on the walls in a similar fashion to panelizing the foil in the unbounded method. The second is to use the method of images where the walls are treated as “mirrors” and imaginary image foils are placed outside the flow domain[28].

Adding panels to the walls is not an attractive option. Adequate representation of the walls with vortex panels requires many more panels than are already used for the foil. This adds significantly to the computer code solution time. Furthermore, the boundary condition at the walls ($\mathbf{V} \cdot \mathbf{n} = 0$) is only satisfied at the individual control points, rather than continuously along the boundary. Another issue with paneling the walls is deciding how far up and down stream to extend the panelization scheme. The required distribution of wall vortex panels varies and is dependent upon many parameters. There is no obvious generalized scheme for implementing a rule based algorithm to determine the minimum required number of panels and their respective spacing on the walls.

A better way to represent the walls in an inviscid flow solver is to use the method of images when the geometry permits. In this case, the foil is located between two parallel walls. Therefore the image geometry is very simple. Using the wall as a symmetry plane, an *image body* is placed on the opposite side of the wall from the actual body. For a vortex situated near one wall the image formulation is simple - - add one image vortex[28]. However with two parallel walls, the image geometry is a

little more complicated. The two parallel walls create an infinite number of reflective planes. This geometry, with the first pair of vortex lattice images, is shown in Figure 2-5. Newman[24] provides a treatment for the similar case of a potential flow point

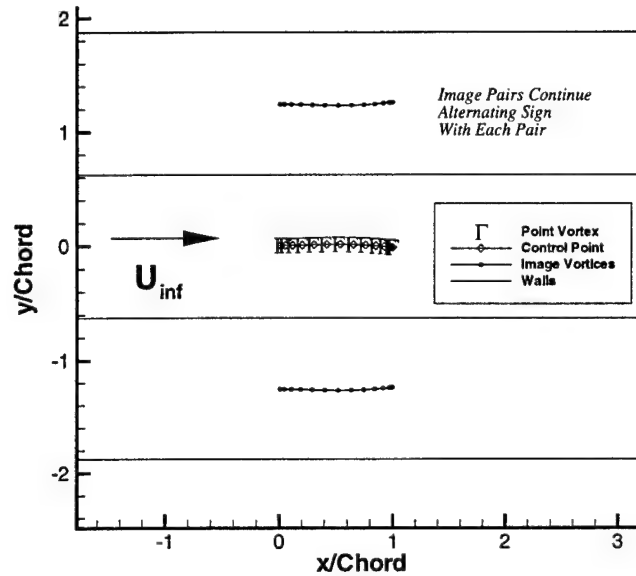


Figure 2-5: Vortex Lattice Near Walls With Images

source situated between two walls located at $y = \pm \frac{1}{2}b$. Implementing this for the case of a distribution of point vortices is analogous. This requires the addition of an infinite array of image vortices at $y = \pm b, \pm 2b, \pm 3b, \dots$, to satisfy the boundary condition at the walls. A closed form solution can be formulated for the infinite array, but this is unnecessary since each subsequent image pair has rapidly diminishing impact on the solution. For all practical purposes, a converged solution can be obtained with a small number of image pairs. This is demonstrated in section 3.5.2. The important thing to remember with this solution scheme is that the image vortices are the exact same strength as the original vortices. Therefore, no additional unknowns are added to equation 2.21. The only additional burden in computation is the addition of the influence of the vortex images to the influence coefficient matrix (\mathbf{A}). Lastly, using

the image representation eliminates the problem of deciding how far up and down stream to extend the walls. Using images, the wall boundary condition is exactly satisfied for $-\infty < x < \infty$.

2.5 Momentum Theory for Lift and Drag Calculations

It has been shown that it is a simple task to calculate the lift and drag by direct integration on the foil surface when using numerical foil analysis tools. However, in water tunnel experiments this is very difficult to do. It is very time consuming and the level of complexity of instrumentation required to take the surface measurements is impractical. The normal methods for measuring lift and drag for foils in the MIT MHL are “bounding box” velocity measurements using laser Doppler velocimetry(LDV)[17]. This is an application of fluid momentum theory. Bounding box calculations are used in this thesis to make comparisons between numerical calculations and experimental measurements. Bounding box contours can also be extracted from RANS solutions as a check of the surface pressure integration results.

2.5.1 Bounding Box Analysis

Bounding box methods are a way of calculating the forces acting on 2-D hydrofoils expressed in terms of integrals of the velocity flow field along a closed contour surrounding the foil[20]. Figure 2-6 is an example of a closed integration contour around a foil. Integrating around the contour C with an outward pointing unit normal vector \mathbf{n} , the momentum flux \mathbf{M} , out of C is:

$$\mathbf{M} = \rho \oint_C \mathbf{q}(\mathbf{q} \cdot \mathbf{n}) ds \quad (2.23)$$

where \mathbf{q} is the fluid flux across C and ρ is the fluid density. The momentum flux through the surface of the foil is zero, since it is by definition a rigid boundary. The momentum flux of the volume of the fluid V enclosed on the outside by C and on the inside by the foil surface is given by equation 2.23. Using Newton’s 3rd law, this

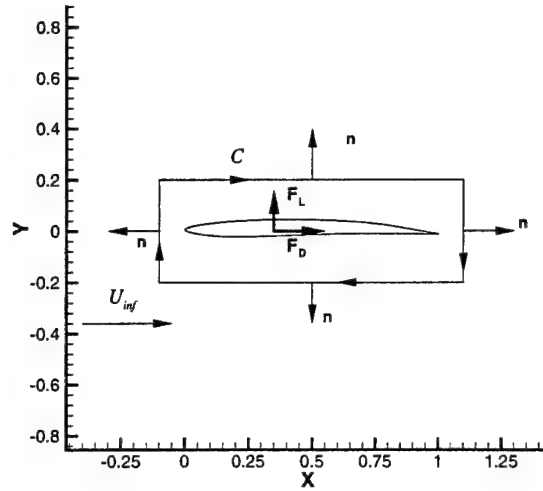


Figure 2-6: Rectangular Contour Around 2-D Hydrofoil

momentum must be balanced by an opposing force \mathbf{F} which is given by:

$$\mathbf{F} = - \oint_C P \mathbf{n} ds + \oint_C \tau \mathbf{t} ds - \mathbf{F}_L - \mathbf{F}_D \quad (2.24)$$

When taking measurements around foils in the MIT MHL with the LDV apparatus, a rectangular contour is frequently chosen. This geometry is also convenient for extracting data from a RANS solver or from vortex lattice field point calculations. The contour is chosen such that it is sufficiently far from the foil so the flow is considered inviscid². Therefore, all the contributions due to the shear terms (τ) are zero and the pressure terms (P) are evaluated using the Bernoulli equation[20].

2.5.2 Drag by Wake Defect

Although in principle, the direct application of equations 2.23 and 2.24 should yield adequate results, this is not always the case for drag measurements. A special treatment must be applied in order to get a better estimate for drag by bounding box methods[20]. First, some ancillary variables should be defined for the x -component

²A minor exception to this is the small region where the wake passes through the downstream side of the box.

of the velocity in different regions of the flow along the right hand side of the contour:

$$\begin{aligned}
u^* &= u, & \text{above the wake} \\
u^* &= u + \Delta u, & \text{in the viscous region of the wake} \\
u^* &= u, & \text{below the wake}
\end{aligned} \tag{2.25}$$

The first order calculation for drag is expressed as:

$$\mathbf{F}_D = \rho U_{inf} \int_{rhs \ of \ C} (u^* - u) dy \tag{2.26}$$

This can be further simplified using equation 2.25, resulting in:

$$\mathbf{F}_D = \rho U_{inf} \int_{across \ wake} (\Delta u) dy \tag{2.27}$$

This equation is correct up to the first order of Δu , as long as the contour which crosses the wake is sufficiently down stream such that $\Delta u \ll U_{inf}$ holds. In the MIT MHL, the optical limits of the tunnel view port window do not allow for this to happen when large foils ($c > 30cm$) are tested. Therefore, a second order correction must be applied to equation 2.27. Typical measurements for bounding boxes around large foils have wake defect velocities which range as low as $0.3U_{inf}$ to $0.5U_{inf}$. Kinnas[20] has shown, using equation 2.24 as a starting point, that the drag including the second order correction is:

$$\mathbf{F}_D = \rho U_{inf} \int_{across \ wake} (\Delta u) dy - \rho \int_{across \ wake} (\Delta u)^2 dy \tag{2.28}$$

These equations for bounding boxes apply to both unbounded and bounded flows. Direct surface integration and bounding box analyses almost never agree exactly. In experiments there is the inherent uncertainty associated with taking a measurement. Additionally, the flow is usually complicated by some unsteady phenomena that occurs such as vortex shedding. In numerical computations there are usually small differences associated with the how the calculation scheme is implemented. When comparing experiments and computer results, the lift measurements usually agree

quite closely. Drag results tend to vary widely. This may well be due to the fact that drag, in comparison to all other flow characteristics, is a small quantity. Accordingly, it is a hard quantity to measure with great precision and consistency.

2.6 Reynolds Averaged Navier Stokes Equations

2.6.1 Derivation of the RANS Equations

One cannot discuss the RANS equations without first reviewing the equations from which they originate. The Navier-Stokes Equations are the equations of motion for a viscous fluid. The derivation can be found in many texts, of which, Sabersky[28] provides a clear step by step formulation. The N-S equations stem from Newton's law of motion and Newton's law of viscous friction. The equations apply to viscous compressible flow with varying viscosity. The N-S equations also apply to turbulent flow. In vector form the equations are:

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{v} + \frac{1}{3}\nu\nabla\Theta + \mathbf{f} \quad (2.29)$$

The Mach number of the cases studied is sufficiently low such that the fluid is incompressible. Therefore the Θ term is zero.

Considering the case of 2-D turbulent flow, the velocity of the fluid can be expressed as a time averaged quantity with a small fluctuating term added to it. Similarly, the pressure can be expressed as a time averaged pressure with a small fluctuating term.

$$\begin{aligned} u &= \bar{u} + u' \\ v &= \bar{v} + v' \\ P &= \bar{P} + P' \end{aligned} \quad (2.30)$$

The fluctuating terms u', v' and P' are defined such that their time averaged value is zero. Substituting equations 2.30 into the N-S equations, yields(written for x component, similar for y):

$$\frac{\partial \bar{u}}{\partial t} + \frac{\partial \bar{u}'}{\partial t} + \frac{\partial \bar{u}^2}{\partial x} + 2\frac{\partial u\bar{u}'}{\partial x} + \frac{\partial u'^2}{\partial x} + \frac{\partial \bar{u}\bar{v}}{\partial y} + \frac{\partial \bar{u}v'}{\partial y} + \frac{\partial u'\bar{v}}{\partial y} + \frac{\partial u'v'}{\partial y}$$

$$= -\frac{1}{\rho} \frac{\partial \bar{P}}{\partial x} - \frac{1}{\rho} \frac{\partial P'}{\partial x} + \nu \left(\frac{\partial^2 \bar{u}}{\partial x^2} + \frac{\partial^2 \bar{u}}{\partial y^2} \right) + \nu \left(\frac{\partial^2 u'}{\partial x^2} + \frac{\partial^2 u'}{\partial y^2} \right). \quad (2.31)$$

If the time averaged is taken for this equation, and the continuity equation is applied for \bar{u} and \bar{v} the equation simplifies to:

$$\frac{D\bar{u}}{Dt} = -\frac{1}{\rho} \frac{\partial \bar{P}}{\partial x} + \frac{1}{\rho} \frac{\partial}{\partial x} \left(\mu \frac{\partial \bar{u}}{\partial x} - \overline{\rho u'^2} \right) + \frac{1}{\rho} \frac{\partial}{\partial y} \left(\mu \frac{\partial \bar{u}}{\partial y} - \overline{\rho u'v'} \right) \quad (2.32)$$

This equation is exactly equation 2.29 with the addition of two terms $\overline{\rho u'^2}$ and $\overline{\rho u'v'}$. The time averaged pressure and velocity satisfy the original N-S equation. In laminar flow these terms are zero. These extra terms are important in turbulent flow. They are commonly called the turbulent stresses or Reynolds stresses.

2.6.2 Turbulence Modeling

With the addition of the Reynolds Stress terms to the N-S equations, more equations must be included in the solution to evaluate the extra unknown terms. As these terms represent the turbulent characteristics of the flow, the extra equations required are usually called turbulence models. The word *model* is used because the formulation of the equations is typically based on empirical studies of turbulent flow. There are several models available for use in solving the RANS equations. Three of the most common derived models are:

- Prandtl mixing length “zero equation” model
- The “two equation” $k - \varepsilon$ model
- Baldwin-Lomax algebraic model

All the above methods employ the concept of eddy viscosity ν_T . The models use empirical relations to establish a value for ν_T which is fed into the RANS equations. The models used by the incompressible RANS solver in this thesis are the $k - \varepsilon$ and the Baldwin-Lomax Model.

The Prandtl model uses a single parameter to determine the eddy viscosity. It is called the mixing length l . The mixing length is the characteristic length of the

mean eddy size which is much smaller than the fluids mean free path. It can also be thought of as the length normal to the main flow where momentum exchange is occurring. The Prandtl mixing model takes form:

$$\nu_T = l^2 \left(\frac{d\bar{u}}{dy} \right) \quad (2.33)$$

This equation directly relates the eddy viscosity to the mean flow. Therefore, no additional unknown terms are added to the problem formulation. The mixing length is usually determined by the experimental results for flows which are similar physically to those being modeled numerically.

A derivation of the $k - \varepsilon$ model is presented in White[30]. This model is based on dissipation. This requires the addition of two equations to the equations governing the fluid flow. One is a turbulent kinetic energy equation and the other is the turbulent dissipation equation. Included in these equations are five empirically derived constants. These constants can be varied so that they can be applied to different types of flows such as wakes and jets. If the solution does not predict the flow reasonably, then the model and its parameters probably do not match the physics of the chosen flow[29]. The model is not intended for flows with high curvature. In lifting surface analysis, there are situations where the flow has a high curvature near the leading edge or un-separated flow adhering to an anti-singing trailing edge. For this reason, the model was not extensively used in this thesis.

The Baldwin-Lomax algebraic model is very attractive because of its computational efficiency and accuracy[29, 30]. The model is appropriate for bodies at *modest* angles of attack where flow separation is minimal or is not expected to occur. This eddy viscosity model defines an effective viscosity ν_e .

$$\nu_e = \nu + \nu_T \quad (2.34)$$

The flow is divided into two regions. Different equations for the calculation of the turbulent eddy viscosity ν_T apply in each region. In the *inner region* the Prandtl

turbulent model(equation 2.33) is used with the van Driest[30] damping model for computing the mixing length. In the *outer region* a different set of equations applies. The eddy viscosity is calculated by[29]:

$$\nu_T = KC_{cp}F_{wake}F_{Kleb}(y) \quad (2.35)$$

In the equation, K , C_{cp} , F_{wake} , and $F_{Kleb}(y)$ are a set of constants and coefficients calculated from the local mean flow characteristics and the distance from a boundary. Equations 2.32(written for both \bar{u} and \bar{v}), 2.33, and 2.35 formulate a set of four equations with four unknown quantities. They are the velocities \bar{u} and \bar{v} , the pressure \bar{P} and the eddy viscosity ν_T .

2.7 Boundary Layer Flows

2.7.1 Characterization of Boundary Layers

A boundary layer, as the name implies, is a thin layer of fluid adjacent to a boundary such as a foil surface or a wall. Within the boundary layer the fluid undergoes a transition from the free stream velocity outside the boundary layer to at rest on the boundary. The majority of the viscous effects are confined to boundary layers.

To employ a RANS solver effectively, it is important to discretize a flow domain so that an appropriate number of fluid elements are contained within the boundary layer to capture the essential viscous effects. In modeling flow around lifting surfaces, failure to adequately capture the flow in the boundary layer typically results in an overestimation of lift coefficient and underestimation of drag. If the boundary layer is missed completely the results tend toward the inviscid potential flow results. Since capturing the boundary layer flow is so important to the accuracy of the RANS solution, it is important to be able to estimate the boundary layer characteristics beforehand. The initial estimates for the boundary layer dimensions are based on flat plate theory which are also suitable for application to lifting surfaces operating at moderate angles of attack.

Many texts provide relevant discussion of boundary layer theory and characteristics, of which Newman[24] and White[30] are well suited to the level of discussion required here.

2.7.2 Laminar Boundary Layer Flows

Consider the case of a flat plate of length l in a two-dimensional fluid domain with uniform flow U . Equation 2.29 reduces to the following:

$$\begin{aligned} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \\ u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= -\frac{1}{\rho} \frac{\partial P}{\partial x} + \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] \\ u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} &= -\frac{1}{\rho} \frac{\partial P}{\partial y} + \nu \left[\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right] \end{aligned} \quad (2.36)$$

The boundary conditions that apply for this flow are that $u \rightarrow U$ just outside the boundary layer and then $u \rightarrow 0$ at the surface through a small distance δ . Within this small distance δ above the plate, the following characteristics are apparent:

$$\begin{aligned} \frac{\partial u}{\partial y} &= 0 \left(\frac{U}{\delta} \right) \\ \frac{\partial u}{\partial x} &= 0 \left(\frac{U}{l} \right) \\ \text{which implies: } \frac{\partial u}{\partial y} &\gg \frac{\partial u}{\partial x} \end{aligned} \quad (2.37)$$

With u and v being 0 on the plate and in light of the relationships in 2.37, 2.36 reduces to the following equations:

$$\begin{aligned} u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= -\frac{1}{\rho} \frac{\partial P}{\partial x} + \nu \frac{\partial^2 u}{\partial y^2} \\ 0 &= \frac{1}{\rho} \frac{\partial P}{\partial y} \end{aligned} \quad (2.38)$$

The following conclusions can be drawn from equation 2.38: First, the pressure change across the distance δ is not significant. Second, the pressure gradient in the x -direction inside the boundary layer is the same as the fluid outside[24]. In the absence of a pressure gradient in the x -direction, these equations can be solved numerically to

yield the Blasius solution for a flat plate. The concept of *boundary layer thickness* is introduced as the distance δ where the velocity $u = 0.99U$. In this formulation, the boundary layer thickness grows with the one-half power of the local Reynolds number(Re_x). The local coordinate x is referenced from the leading edge of the plate.

$$\delta(x) = 4.9 \left(\frac{\nu x}{U} \right)^{\frac{1}{2}} \quad (2.39)$$

Another quantity that can be computed is the *displacement thickness* or δ^* . This defines an effective thickening of the body due to viscous effects which corresponds to a lost quantity of fluid flux within the boundary layer.

$$\delta^*(x) = \int_0^\infty \left(1 - \frac{u}{U} \right) dy \approx 1.72 \left(\frac{\nu x}{U} \right)^{\frac{1}{2}} \quad (2.40)$$

Associated with the δ^* flux reduction, there is a corresponding loss in fluid momentum and a *momentum thickness* which is calculated by:

$$\theta(x) = \int_0^\infty \frac{u}{U} \left(1 - \frac{u}{U} \right) dy \approx 0.664 \left(\frac{\nu x}{U} \right)^{\frac{1}{2}} \quad (2.41)$$

To calculate the drag on the flat plate due to viscous forces, the shear stress(τ_{xy}) at the plate must be determined.

$$\tau_{xy}(x) = \mu \left(\frac{\partial u}{\partial y} \right)_{y=0} \approx 0.332 \rho U^2 Re_x^{-\frac{1}{2}} \quad (2.42)$$

Once the shear stress is known along the plate, the total frictional drag can be computed using equation 2.13.

All of the above results for a flat plate can be extended to a general 2-D body without loss of validity, provided that the radius of curvature is much larger than the boundary layer thickness.

The pressure gradient in the x -direction does affect the boundary layer. The boundary layer will become separated from the body at a point when the following conditions are met: the shear stress is zero; upstream of that point the tangential velocity is positive; and downstream of that point it is negative. A separated region is characterized by the streamlines breaking away from the body downstream. A separated wake is an area of relatively high vorticity and low pressure.

2.7.3 Turbulent Boundary Layers

If the Reynolds number is sufficiently high, the laminar boundary layer will *transition* or change to a turbulent boundary layer at some distance aft of the leading edge of the foil. In the laminar boundary layer the velocity profile is smooth and regular and predictable. At some point, the flow becomes disturbed and unstable. Transition occurs. After this, the boundary layer will typically remain turbulent along the remainder of the surface. The boundary layer remains relatively thin. The exact transition point can only be estimated and its location is dependent on Reynolds number and local roughness of the surface. Typically in experimental practice, turbulent transition is forced at a prescribed location near the leading edge of the foil. This is normally accomplished by the addition of surface irregularities such as raised bumps at the desired point of transition. Near the surface, inside the boundary layer, there is a viscous sub-layer that is small compared to the overall boundary layer thickness. The region between the viscous sub-layer and the outer fluid is called the *turbulent core*.

The fluid flow in a turbulent boundary layer is more complicated than the laminar case. Mixing occurs at the interface between the free stream and the boundary layer. The basic momentum equations that apply to the laminar case apply to the turbulent boundary layer as well. At the surface the shear stress boundary condition still applies ($\tau_{ns} \propto \text{Velocity gradient}$). The flow in the turbulent core and at the interface with the outer fluid is complicated. The assumption that the slope of the velocity profile provides an adequate value for shear stress (equation 2.42) is no longer valid[28]. As a result, empirical derivations of turbulent shear stress near a wall are typically used. One of the most common formulations is the $1/7^{th}$ power law approximation:

$$\frac{u}{u_\tau} = 8.7 \left(\frac{yu_\tau}{\nu} \right)^{1/7} \quad (2.43)$$

After making the substitutions $u_\tau = \sqrt{\tau/\rho}$, $u = U_{inf}$ at $y = \delta$, the following equations

result for shear stress at the surface(τ_{ns}) and turbulent boundary layer thickness(δ):

$$\tau_{ns} = 0.0463 \left(\frac{a}{Re_x} \right)^{1/5} \rho U_{inf}^2 \quad (2.44)$$

$$\delta = 0.058 \left(\frac{\nu}{U_{inf}} \right)^{1/5} \left(\frac{x}{a} \right)^{4/5} \quad (2.45)$$

where the term a is determined by the computation of the integral:

$$a = \int_0^1 \left(1 - \frac{u}{U_{inf}} \right) \frac{u}{U_{inf}} d \left(\frac{y}{\delta} \right) \quad (2.46)$$

These equations are included as they were used as a check for validating the quantities determined from the RANS results. A more complete discussion of turbulent layers and the development of the associated equations is contained in Sabersky[28].

2.7.4 The y^+ Parameter

Finite element cell spacing near any no-slip boundary needs to be much smaller than the boundary layer thickness δ . This is necessary to accurately capture the velocity profile completely through the boundary layer to the no-slip surface. The distance that the close-in grid points are measured from the no-slip surface are measured in multiples of y^+ where:

$$y^+ = u_\tau \frac{y}{\nu} \quad (2.47)$$

and

$$u_\tau = \sqrt{\frac{\tau_w}{\rho U^2}} \quad (2.48)$$

Based on Anderson[2], Black[6] and present RANS methodology in the MIT-MHL, y^+ values for the first three cells off the no-slip surface should be < 1 , ≤ 2 , and ≤ 4 respectively. Additionally, there should be another 10 to 15 cells spaced from $y^+ \approx 4$ to $y^+ \approx 100$. It should be noted that failure to adhere to this tight spacing criteria can result in large errors.

2.7.5 The Law of the Wall

The *Law of the Wall* is a commonly used formulation for determining the velocity profile characteristics near a wall[28]. The main assertion is that the velocity profile(u)

is determined by the conditions at the wall(τ_{ns}), the fluid properties(ρ, ν) and the distance from the wall(y). Using the Buckingham Pi theorem, two dimensionless parameters can be formulated from these quantities[23]. One is a non-dimensional velocity and the other a non-dimensional distance. There is an implied simple functional relationship between the two parameters:

$$\frac{\bar{u}}{u_\tau} = f\left(\frac{yu_\tau}{\nu}\right) \quad (2.49)$$

As a matter of convenience, the terms in equation 2.49 are often replaced by $u^+ = f(y^+)$. Several approximate formulas have been deduced based on experimental data. Of those available, the Spalding formula is quite suitable because it fits experimental data well from $y^+ \approx 100$ all the way down to the surface[30].

$$y^+ = u^+ + e^{-\kappa B} \left[e^{\kappa u^+} - 1 - \kappa u^+ - \frac{(\kappa u^+)^2}{2} - \frac{(\kappa u^+)^3}{6} \right] \quad (2.50)$$

In the above equation, values for the constants κ and B are 0.41 and 5.0 respectively. This formula is plotted in Figure 2-7. The Spalding formula for the *Law of the Wall* is

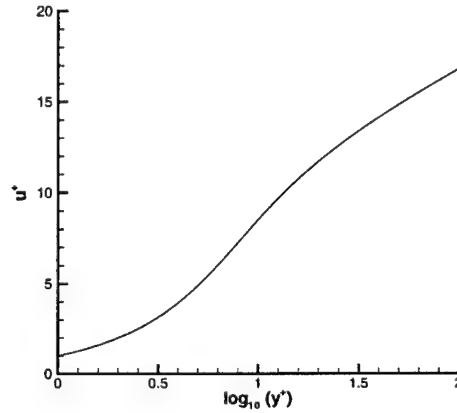


Figure 2-7: Spalding Formula for *Law of the Wall*

used for validation of RANS calculations in selected cases to ensure that the velocity profiles close to the wall are correctly captured. One measure of the quality of a numerical finite element grid is how well the calculated boundary layer profiles agree

with the Spalding formula. It can be concluded that if the grid spacing conforms to the y^+ spacing criteria and the computed boundary layer profiles conform to the *Law of the Wall*, then the numerical grid is an adequate model for the flow.

Chapter 3

Numerical Methods Development

3.1 Overview

Several 2-D foil analysis computer codes are employed in this study. The majority of the analysis was conducted using the RANS flow solver DTNS2D[29]. Other codes used are PAN2D[22] and XFOIL[11].

DTNS2D is a generalized 2-D domain incompressible RANS fluid flow solver developed at the U.S. Naval Surface Warfare Center, Carderock Division. DTNS2D uses a finite volume finite difference formulation. In a finite difference solution method, the flow domain is divided into a set of discrete control volumes. Within each element, the governing equations of the flow domain are solved in their integral form. It is capable of modeling foils in bounded and unbounded domains. It should be noted that there are several RANS computer codes that could be applied to the cases studied here. However, DTNS2D was a readily available code within the MIT-MHL that required no modification for employment in this thesis. It has been used previously for analyzing lifting surfaces with results that agree with experimental data[25]. Therefore, it was deemed adequate for this study as well.

PAN2D and XFOIL use potential flow panel methods for calculating the inviscid flow characteristics around a foil in an unbounded 2-D domain. The solution is then coupled with an integral boundary layer method to determine the viscous properties of the flow[15]. The two panel methods were used to validate DTNS2D results for

unbounded flows. The two panel methods are currently not capable of modeling foils bounded by tunnel walls.

In this chapter the development process for the FIT2D computer code that provides the fluid geometry input files for the DTNS2D solver is presented. This involves discretizing the domain into a mesh of four sided polygons finite volume elements. Grid generation schemes and methodology are discussed. As a subsection of FIT2D, a 2-D vortex lattice computer code[18] was implemented to obtain an inviscid solution for the lift coefficient of a foil in unbounded and bounded flows. The vortex lattice solution is also used to grow a dividing streamline downstream from the trailing edge to define a RANS zonal grid boundary.

3.2 Overview of Grid Generation

3.2.1 Fundamental Concepts

The overlying governing directive of grid generation is simple: discretize the domain into a sufficient number of elements such that the essential physics of the flow is captured. The implementation of this is difficult. Figure 3-1 is a representation of a discretized fluid domain surrounding a hydrofoil inside a tunnel. It is important to notice that the grid spacing is not uniform. The size of the individual elements must be varied depending on the expected local characteristics of the flow. For example, near a wall or on the surface of the foil, the vertical spacing must be quite small in order to accurately capture the velocity gradient within the boundary layer. For hydrofoils it is also important to *cluster* elements near the leading edge to capture the strong pressure gradients associated with the flow stagnation point. At the trailing edge, clustering is used where separated flow is expected. Additionally, the wake behind the foil has steep velocity gradients that slowly dissipate with the flow. The clustering scheme is advantageous if the small elements are located within the wake region. Conversely, there are regions in the flow where very little is occurring. Velocity

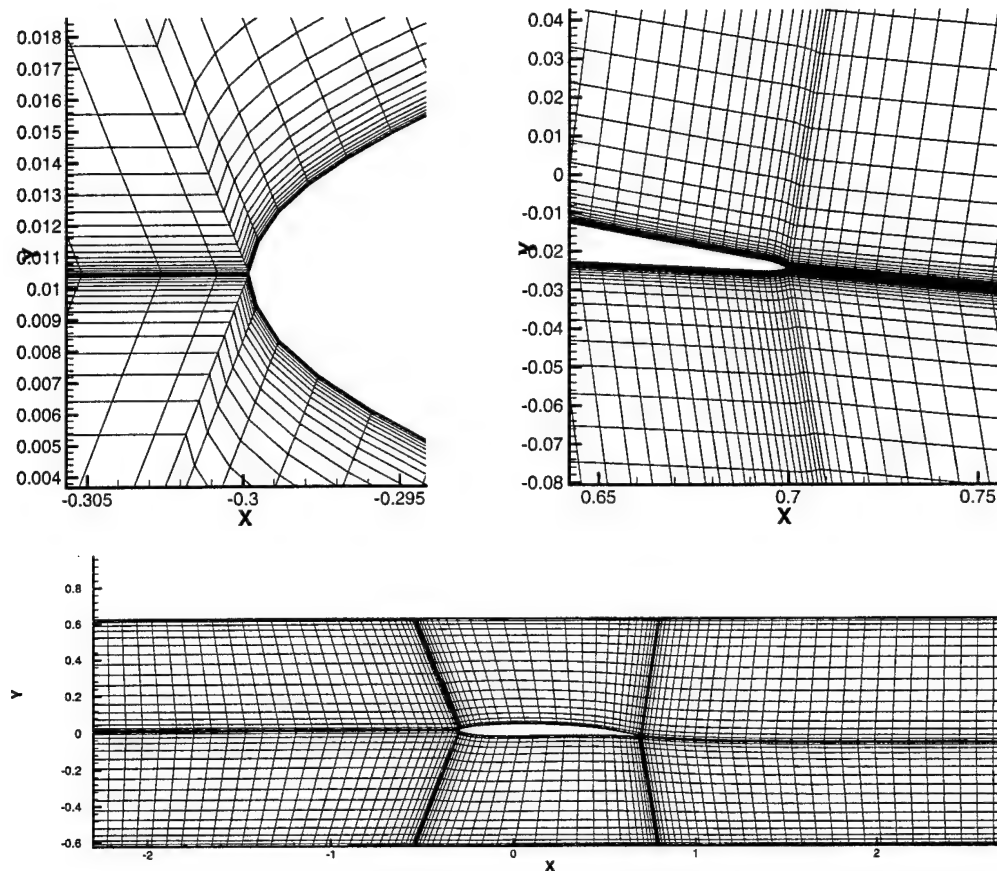


Figure 3-1: Sample of the discretized flow domain of a foil in a water tunnel. (Only 1/3 of the total grid lines are reproduced in the lower frame)

and pressure gradients are near zero. In these areas, there is no reason to have finely spaced cells.

In Figure 3-1, the element dimensions vary throughout the domain. The change in size from largest to smallest element is approximately four orders of magnitude. This sample representation requires 41,238 elements. In a grid representation of an unbounded flow domain around a foil, the element dimensions can range as much as six orders of magnitude. Of course one could simply find the smallest dimensional requirement for an element (eg. foil leading edge) and set all elements to that size and capture the essential physics of the flow. The problem is, for the same sample above,

the grid would require 6×10^{10} elements! In a typical numerical solver, the solution time T is proportional to the number of elements squared ($T \propto n^2$). For the same results, the processor time to obtain a solution has grown by a factor of 10^{12} . Clearly this is not a practical solution. So, there are two conflicting requirements which must be met in order to produce results which are both practical and accurate. The first is to maximize the number of elements to capture all the pertinent physics. The other is minimize the number of elements to obtain the minimum solution time yet also obtain correct results. To satisfy both of these driving forces, the element sizes must vary throughout the domain in order to capture the local effects and minimize the overall number of elements.

3.2.2 Geometric Limitations of DTNS2D

Before proceeding with the description of the FIT2D program, it is important to specify the geometric limitations for the DTNS2D input files. In the DTNS2D program the individual elements are grouped into zones. There can be any number of zones in DTNS2D. Figure 3-2 is a typical zone for DTNS2D. A zone consists of four sides. The sides can consist of any shape to enclose the zone so long as the four sides together comprise a simple connected region. A simple closed region has the property that an arbitrary closed curve lying in the region can be shrunk continuously to a point in the region without passing outside of the region[14]. Each pair of opposite sides must have the same number of elements along the side. Adjacent sides are not required to have equal number of elements. The elements within each zone cannot overlap any other elements. Every element within the zone must have four sides of finite non-zero length. Each zone side has a defined boundary condition (*eg.* no-slip, tangent, free stream, continuity of velocity gradient or pressure gradient). The boundary condition along a side can not change. Adjacent sides can have different boundary conditions. An example of a DTNS2D zone model is presented in Figure 3-3. This type of model is frequently called an "H" grid.

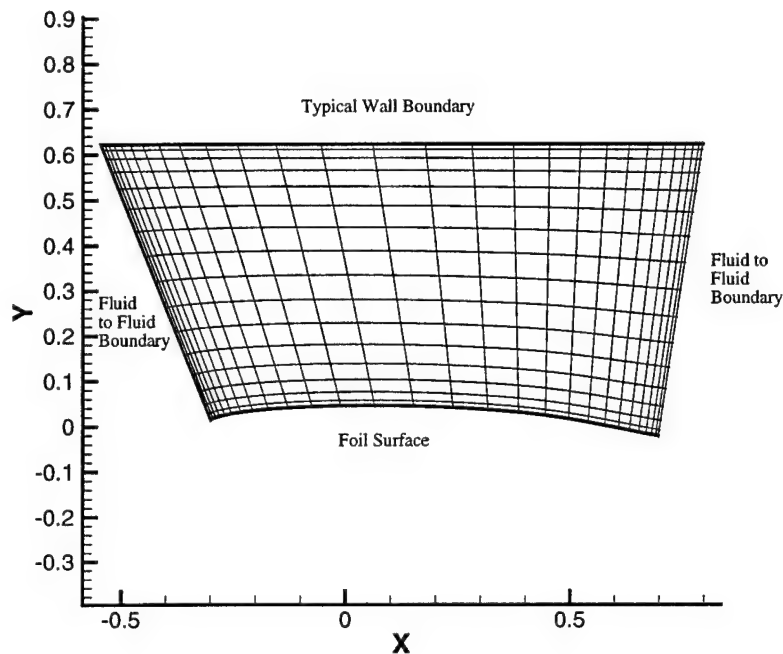


Figure 3-2: Sample geometry of a DTNS2D zone. (Only 1/3 of the total grid lines are reproduced)

3.3 Development of the Computer Code FIT2D

3.3.1 Functional Requirements

Recent research projects in the MIT-MHL water tunnel have demonstrated a requirement to use a RANS computer code to model flow around hydrofoils[19]. Some experiments on foils with unique camber distributions yielded interesting results. It is anticipated that the RANS output could help provide some insight into the results that were being observed.

Before the development of FIT2D, generating the DTNS2D RANS solver input files was time consuming and tedious. A requirement was established for the development of a computer program that could quickly generate the flow domain and RANS input files for any 2-D hydrofoil that could conceivably be tested in the MIT-MHL. This stated need was translated into the following set of functional requirements or program

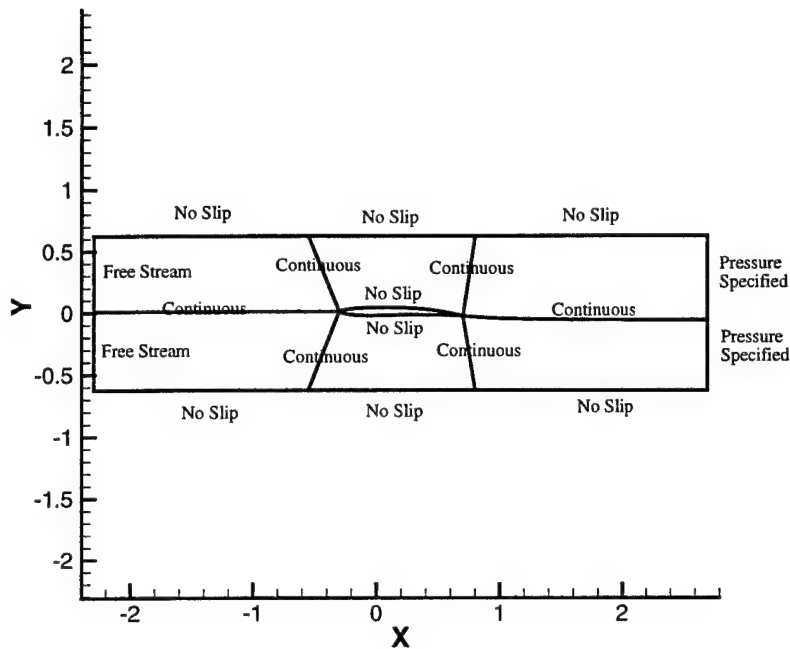


Figure 3-3: Typical DTNS2D zonal boundaries for a foil in a tunnel

objectives:

1. Run from within a simple user interface.
2. Support basic inputs such as angle of attack, domain extents, and locations of walls if any.
3. Allow the user to specify grid spacing in key areas such as the leading edge and trailing edge and inside the boundary layer.
4. Evaluate user spacing of elements inside the boundary layer and compare against flat plate calculations. Make a recommendation for any changes.
5. Read in x, y foil offsets. Spline the offsets and locate the leading edge.
6. Rotate the foil to user prescribed angle of attack.
7. Establish DTNS zone boundaries for a six zone "H" grid.
8. Allow for the zone boundary going downstream from the trailing edge to follow an established wake or a first guess wake from a vortex lattice calculation.
9. Generate all interior grid points for the mesh using isoparametric interpolation.
10. Write datafiles for the program TECPLOT containing a representation of the grid that can be viewed for correctness.
11. Write datafiles for the INMESH grid smoothing program.

3.3.2 Structure of FIT2D

FIT2D is written using the FORTRAN 77 code standard. It is executed interactively from a UNIX command window. The typical setup when running FIT2D is to have the data input file or *control* file displayed in an EMACS window(or any other editor), and a TECPLOT graphics window for viewing meshes. The three window configuration was chosen in favor of writing a dedicated X-Windows application. A UNIX command window, EMACS and TECPLOT provide an environment to which many people are already accustomed. After setting up the initial foil offset file(*fname.foil*) and control file(*fname.ctrl*), the user loops through the following basic procedure:

1. Initialize FIT2D
2. Follow prompts to generate a grid.
3. View the grid in the graphics window.
4. If the grid is satisfactory, confirm permission to write INMESH files.
5. If the grid is unsatisfactory, make changes to *fname.ctrl* file and reinitialize FIT2D.

Once satisfactory results have been achieved with FIT2D, the program INMESH is run to smooth the grid. This process is described in greater detail in section 3.6. Once the grid has been generated and smoothed, the geometry is ready for input to the RANS solver. A complete listing of the Program FIT2D is contained in Appendix A.

3.3.3 Selecting Required Input

Sample input files for FIT2D are contained in Appendix B. With the exception of input/output control, and boundary layer resolution changes, all FIT2D input is in batch form. The control file contains all of the data for describing the flow domain and how the mesh of elements will be distributed. The required input stems from the ideas presented in section 3.2.1. In the interest of making grid generation a somewhat more mechanical and less artistic process, it was decided early on to limit user input to

Table 3.1: Batch Input Variables for FIT2D

FOILGEO	<i>fname</i> .foil geometry file name, located in same directory
AOA	Angle of Attack in degrees
Xpiv	distance x/c of pivot point from leading edge
Ypiv	distance y/c of pivot point from nose-tail line
TUNPARAM	foil scale parameter c /tunnel width
USL	upstream limit of flow domain in chord lengths from L.E.
DSL	downstream limit of flow domain in chord lengths from T.E.
PHI1	forward rake distance of vertical zone lines from leading edge
PHI2	aft rake distance of vertical zone lines from trailing edge
NUS	number of elements in x -direction upstream
NDS	number of elements in x -direction downstream
NVS	number of elements in y -direction above and below foil
NTOP	number of elements along chord on top surface of foil
NBOT	number of elements along chord on bottom surface of foil
RESLE	Element width in chord lengths at leading edge
RESMID	Element width in chord lengths at mid chord
RESTE	Element width in chord lengths at trailing edge
RESWALL	Element height in chord lengths at the walls
RESBL	Element height in chord lengths on the foil surface
PACK	fraction of NVS elements targeted within boundary layer
$Re\#$	chord based Reynolds number
MESHFILE	INMESH input file name
RESTFILE	INMESH restart file name
NUMIT	number of INMESH iterations
TOL	convergence tolerance limit for INMESH
NRANSWK	wake adaption flag: < 0:straight line, 0: use VLM, > 0 use RANS data
RANSWKPTS	x, y coordinates of the RANS wake line data

a few key parameters to guide the program. Within the control file, there are fifteen essential parameters for grid generation that the user must specify. Additionally, there are some required administrative inputs. In the control file, each line of input is preceded by a descriptor line. The key inputs and their functions are summarized in table 3.1. Most of the parameters are intuitive. However, two parameters in particular need further explanation. The PHI1 and PHI2 rake angles are required in the mesh so that elements at the leading and trailing edge do not compress to a zero volume. An example of a problem leading edge is one that is circular with a large radius. If the zone boundary extended vertically from the leading edge, then the cells to the right of the boundary and above the leading edge would be skewed almost ninety degrees. In other words, cell volume is zero making it a singularity point. Referring

to Figures 3-1 and 3-3, one can see that the zonal boundaries are raked. The visible effect is that the cells retain a *rectangular* profile as they wrap around the leading edge. The upstream cells experience a minimal volume compression. But, the overall effect is to improve the consistency of the cell volumes. The raking capability was added at the trailing edge to accommodate highly curved anti-singing trailing edges.

3.3.4 Splining and Element Distribution Methods

All of the zonal boundary lines are splined parametrically in arc length using a cubic splining routine. The cubic coefficients are determined and stored for later use. Cubic splining in arc length was not required for any of the straight line boundaries. It was used anyway because it allows for future growth of the computer code to easily accept zonal boundary lines of arbitrary shape. An example of this is using a "C" shaped zone that wraps around the foil.

The user specifies how many points are to be distributed along each boundary as well as the endpoint element dimension. Several schemes were explored to distribute the remaining points along the boundaries. The rate of change in cell dimensions should be relatively uniform along each boundary. Uniform spacing obviously is not a candidate. Cosine spacing was investigated, but it proved inadequate because the growth rate of cells near the areas of key interest was too fast. Ultimately, two spacing methods proved preferential for distributing elements. One spacing method is for *horizontal* zone lines. The other is for *vertical* zone lines.

The *horizontal* lines define the tunnel walls, foil surface, upstream zone lines, and the wake. Along horizontal lines a parametric cubic distribution routine is employed. This routine was developed by Black[5] for distributing points on axisymmetric bodies and ducts. The program has been converted to a subroutine for inclusion in FIT2D. It has proven to be adequate for a variety of geometries.

The spacing of elements on the vertical lines is somewhat more difficult. Vertical lines define how the elements are distributed normal to the foil and the tunnel walls.

There are two driving factors. One is to place an adequate number of very small elements close to the foil and walls to accurately capture the boundary layer profile. The other is to have much larger elements outside the boundary layer to minimize the total number. The user sets the spacing by specifying values for the variables: RESBL, RESWALL, NVS, and PACK. RESBL and RESWALL should be set using the y^+ method of section 2.7.4. Poor selection of RESBL and RESWALL (*i.e.* too large) will yield poor results and the boundary layer characteristics will be missed. The next trade-off is with NVS and PACK. It is always easy to increase number of points NVS, but the solution time suffers. PACK dictates how many of the total points NVS will be allocated to the boundary layer.

Given that the steepest velocity gradients occur nearest to the surface, the following spacing routine for vertical lines has been chosen. Set first element height at RESBL or RESWALL. Then geometrically grow the cells by a factor of 1.25 until the number of elements corresponding to PACK are used. Lastly, distribute the remainder of the cells using the cubic distribution routine.

3.3.5 Defining Domain Boundaries

Defining the length limits of the discretized domain was approached in several ways. For analyzing cases for foils in the tunnel, the actual physical limits of the tunnel were used to constrain the upstream and downstream limits of the gridded geometry. In the initial formulation of FIT2D, it was not foreseen that there would be a large need for analyzing unbounded flow around foils using the RANS code. However, as the research process progressed, more and more cases were analyzed in unbounded domains.

The bounded case methodology.

Appendix C shows the configuration of the MIT-MHL water tunnel test facility. Figure C-2 highlights the specifics of the test section where foils are mounted for

measurement. Once the scale of a subject foil is known with respect to the tunnel dimensions, the wall locations and upstream and downstream limits can be calculated. These parameters are supplied to the input files as: USL, DSL, and TUNPARAM. FIT2D then places all boundaries at the proper location to replicate the tunnel geometry. This methodology proved to be adequate. Results obtained agree well with MIT-MHL tunnel experimental results for lift, drag and wake profiles.

The unbounded case methodology.

For unbounded analysis the domain limits were extended. The boundary condition at the walls was changed from no-slip to tangent and the walls were moved *far* away from the foil. A convergence study was conducted using the vortex lattice subroutine to determine how far to locate the walls from the foil such that the flow around the foil could be considered unbounded. A distance of five chord lengths was established for the walls. At this distance, the difference between the lift coefficient calculated by VLM with images and without is approximately 0.1%. Pressure distributions on a foil surface were compared against results from the computer code PAN2D and the differences were negligible. The flow domain was extended upstream and downstream as well. Five chord lengths was judged suitable. Observation of pressure and velocity contours from the RANS outputs showed good uniformity at the inflow and outflow ends of the domain.

3.3.6 Isoparametric Interpolation

Once the zone boundaries have been identified and the cell spacing on the boundary is established, the initial location for the interior cell vertices must be established. This is a critical step in the grid generation process. The challenge is to locate all the interior points such that none of the quadrilaterals overlap and all the points are physically interior to the zone boundary. For a simple geometric shape such as a rectangle or a trapezoid the task is not overly demanding. Difficulties arise when

adjacent boundaries are curved or highly skewed. To overcome these difficulties, a methodology was devised wherein the arbitrary zone boundary geometry is parametrically mapped to a unit square. The interior points are determined using simple line geometry. Then, the interior points are mapped back out using the inverse mapping function. This process is schematically shown in Figure 3-4. Numerically, this is accomplished using the subroutine INTERP which is a variation of *isoparametric interpolation* adapted from Bathe's method for finite element formulation[4].

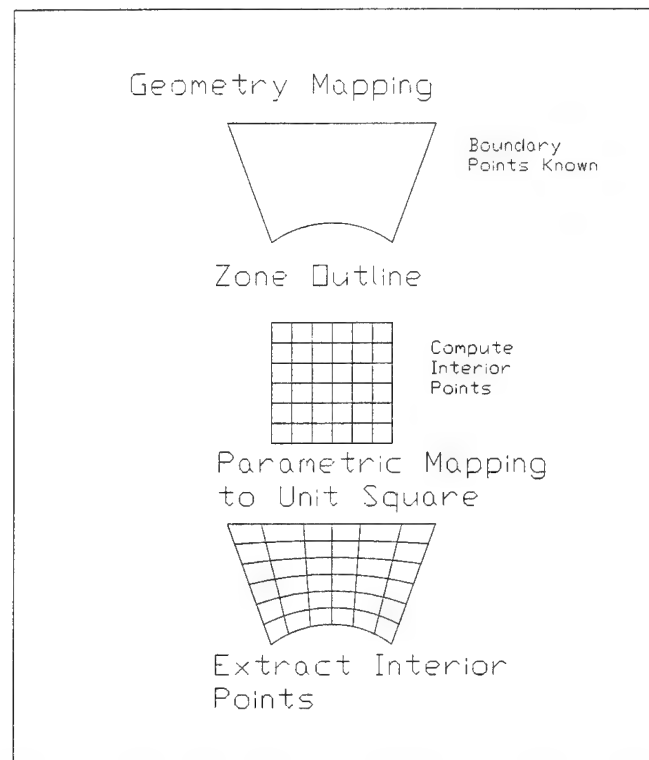


Figure 3-4: Isoparametric Mapping Schematic of Interior Zonal Points

3.4 FIT2D Output

3.4.1 Tecplot Files

Two ASCII plotting data files are generated by FIT2D. These files conform the data format required by the Amtec program TECPLOT version 7.

The first file is *rotate.plt*. It contains the following:

- input geometry rotated to the desired angle of attack
- an overlay of splined foil surface points used for foil surface zone boundaries
- a horizontal reference line that passes through the x, y location of the foil pivot point.

The purpose of this file is to validate the input foil geometry and to ensure that the pivot point and angle of attack have been specified properly. A sample of a figure created with *rotate.plt* is contained in Figure 3-5.

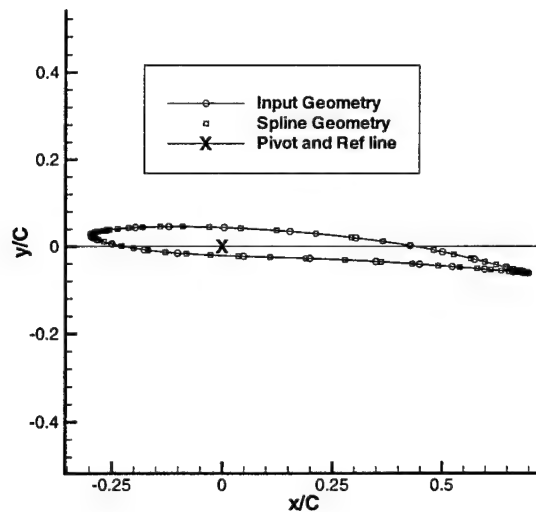


Figure 3-5: Display of *rotate.plt* data

The second file is *mesh.plt*. This file contains the complete grid geometry for all of the zones. The purpose of this file is to allow the user to examine the grid computed by FIT2D to ensure that the desired spacing has been obtained and that the grid lines are smooth. Examples of plots generated with this file are presented in Figures 3-1, 3-2, and 3-3.

3.4.2 INMESH Input Files

Once a satisfactory grid geometry has been obtained, the main output files from FIT2D are written. These files contain all the data required for running the computer

code INMESH(section 3.6). They are called the *input* and *restart* files. The input file is an ASCII data file which defines the zonal boundaries and how the cell points are distributed on the boundaries. The restart file is a binary formatted data file which contains all the interior zone point coordinates in addition to the zonal boundary points.

3.5 Adjustment of Zone Boundaries in the Wake

In section 2.4 the importance of aligning the grid zone boundaries aft of the foil to the convected wake was discussed. A special treatment was required to adapt the fluid mesh geometry to the expected wake line. Two methods were developed for wake adaption. The first is a vortex lattice method which finds the inviscid wake streamline that convects downstream from the trailing edge. A beneficial by-product of the vortex lattice solution is that a good approximation of the inviscid lift coefficient is obtained. The second method uses an initial RANS solution to extract a viscous wake streamline.

3.5.1 Implementation of the ADAPT Subroutine

A vortex lattice subroutine was implemented in the program which extracts the mean camber line of the subject foil in order to solve for the required circulation distribution. Once the circulation distribution is known, the wake can be calculated. This is accomplished by extracting the field point velocity at the trailing edge and then incrementally marching in the direction of the field point velocity vector while extracting the field point velocity at each increment. This technique is visually demonstrated in Figure 3-6. The numerical implementation of the vortex lattice method in the subroutine ADAPT is a direct extension from the theory presented in section 2.4.

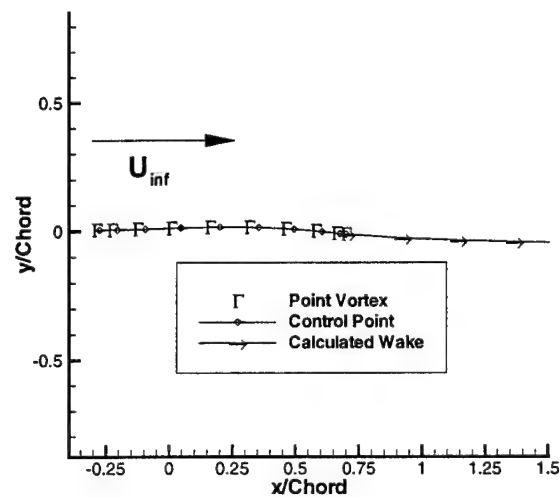


Figure 3-6: Vortex Lattice Geometry with Extracted Wake Line

3.5.2 Adding Image Vortices

The ADAPT subroutine was initially constructed without image vortices which would account for the wall effect. This proved satisfactory for analysis of unbounded flow around foils where there was little or no separation. However, the wake from the unbounded VLM solution tended to continue to convect away from the true wake when walls were present. Initially, it was thought that a correction could be applied to the unbounded VLM which would straighten out the wake. Several attempts were undertaken to develop schemes which redirected the velocity vector based on wall proximity and inviscid lift coefficient. This was made to work well for a single foil at a limited range of angle of attack. But the method was not robust for universal application and would require perturbing the parameters with each application. The addition of vortex images was then explored. Following the development in section 2.4.3, pairs of image vortex lines were added to the ADAPT routine. The final results demonstrating the improved wake tracking are shown in Figure 3-7. The inset panel clearly shows how the unbounded solution continues to convect downward downstream while the wake of the VLM with images eventually becomes parallel to

the tunnel walls.

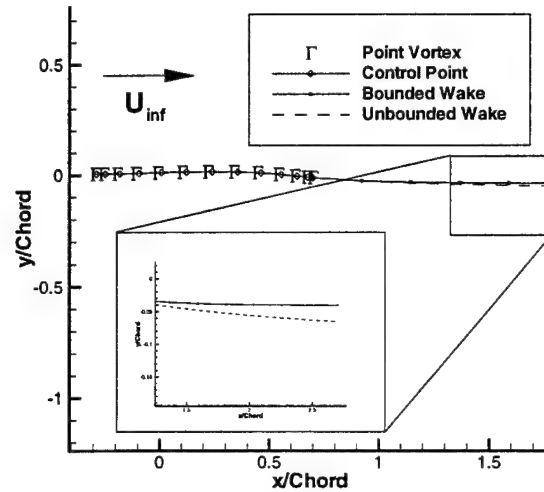


Figure 3-7: Comparison of wakes for VLM(unbounded) to VLM with Images(bounded)

3.5.3 Convergence of the Vortex Lattice Method

Studying the convergence criteria of the VLM with images is a two-dimensional problem. The goal of the convergence study is to determine a minimum number of components that must be modeled in the computer code in order to obtain a reasonable solution. First, one has to consider what is an adequate number of vortex panels to demonstrate convergence for the lift coefficient. Next, one must consider how many pairs of images must be included such that the addition of more image pairs has no effect on the solution. The results of the convergence study are presented in Figures 3-8 and 3-9. Based upon the trend of the two curves, the final configuration selected for the VLM solution employs 40 vortex panels and 16 sets of image pairs. This yields an error in inviscid lift coefficient of less than 0.04%. The percent error of lift coefficient is defined using the single precision numerically converged value for the lift coefficient with 640 vortex panels and 128 sets of image pairs as the reference.

Structuring the grid zones aft of the foil using the VLM wake was useful in cases

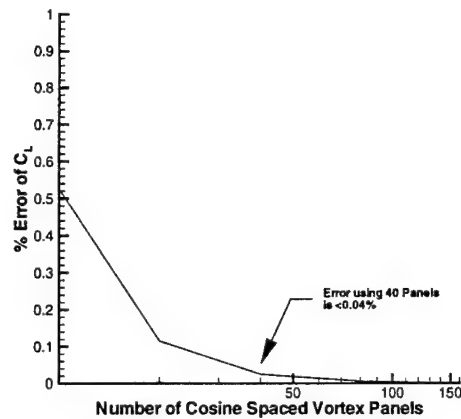


Figure 3-8: Vortex Lattice Panel Convergence Curve

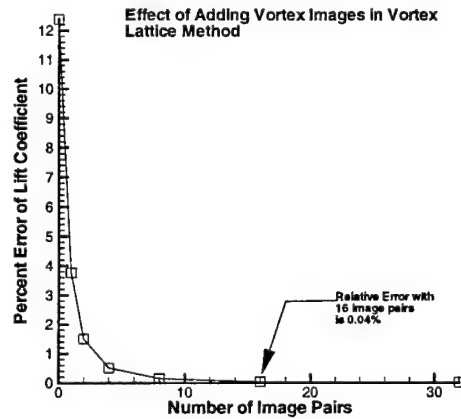


Figure 3-9: Vortex Lattice Image Pair Convergence Curve

where tunnel walls are present and there is little or no separation occurring at the trailing edge. The VLM wake was not a good approximation of the true wake for foils that had extreme camber distributions at the trailing edge or that had separated flow. In these cases the viscous effects on the lift coefficient become more important and the inviscid solution has less validity. When the viscous effects start to become significant, the VLM wake overshoots the true wake in the same manner that the VLM without images overshoots in the bounded flow case.

3.5.4 Adapting Wake Zone Boundaries Using RANS Output

Despite refinements to the vortex lattice method, a significant difference persisted between the wake predicted by the VLM wake and the wake profile obtained using the RANS solution. Therefore, a method was developed to extract the wake data from an initial RANS solution based upon the the vortex lattice wake estimate. This is accomplished using the following steps:

1. Generate grid using the VLM computed wake.
2. Run the RANS solver for a few thousand iterations.
3. Extract data for the wake centerline streamline using TECPLOT.
4. Condition the data with the computer code GETWAKE.
5. Update the *fname.ctrl* file with the wake data.
6. Generate a new grid using the RANS wake data.
7. Run the RANS solver to convergence.

The program GETWAKE is listed in Appendix D. It is a simple conditioner that extracts a representative truncated set of datapoints from the TECPLOT streamline data. There is a major drawback to this wake correction process. It is very time consuming because the RANS solver must be used iteratively. It should be restated that this is only necessary in cases where separation has occurred near the trailing edge or the camber distribution on the aft part of the foil is unusual or both. A comparison between the two wake adaption schemes is shown in Figure 3-10.

3.6 Using a Poisson Solver to Refine Grid

3.6.1 The Program INMESH

The computer code INMESH is used to refine the grid geometry output from FIT2D and generate the geometry input files for DTNS2D[8]. INMESH is a robust grid generation and refinement tool that has been used in a variety of fluid mesh geometry applications. INMESH uses an elliptic solver to approximate the Poisson equation over a prescribed domain with specified boundary values. The calculated streamlines

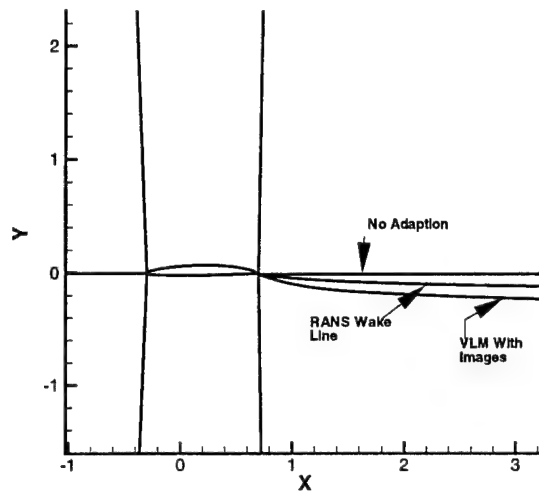


Figure 3-10: Sample Comparison of VLM Wake and RANS Wake Zone Adaption

and equipotential lines are used to formulate the grid. The forcing term in the Poisson equation is used to cluster the grid near surface boundaries and to ensure orthogonality.

INMESH uses the Successive-Over-Relaxation(SOR) method to solve the system of equations. The discretized equations are solved by iteratively sweeping through the domain and shifting the values at the nodes until convergence is achieved. A superb analogy for this process was described by Black[6]:

The best analogy to this solution process would be to consider the initial guess at a grid to be a bedspring system where each node has four springs attaching it to its neighbors. The spacing of the nodes on the boundaries is controlled by the user and clustering could be seen as variable strength springs. The SOR method releases each node and allows it to move to its 'natural' position. By iteratively sweeping through the domain, the nodes will eventually stabilize to the final grid.

INMESH is capable of commencing the grid generation process with only the boundary points specified. Then begins starts with a set of uniformly spaced interior points and starts the *bed spring* process. By employing the isoparametric spacing routine in FIT2D, a reasonable first guess at the location of all the interior points is obtained. These points are provided to INMESH in the binary restart file. A

significant benefit from doing this is that the INMESH code requires several thousand fewer iterations to achieve the same convergence tolerance. In general, a higher relaxation coefficient can also be used. Figure 3-11 demonstrates the results of grid refinement using the program INMESH.

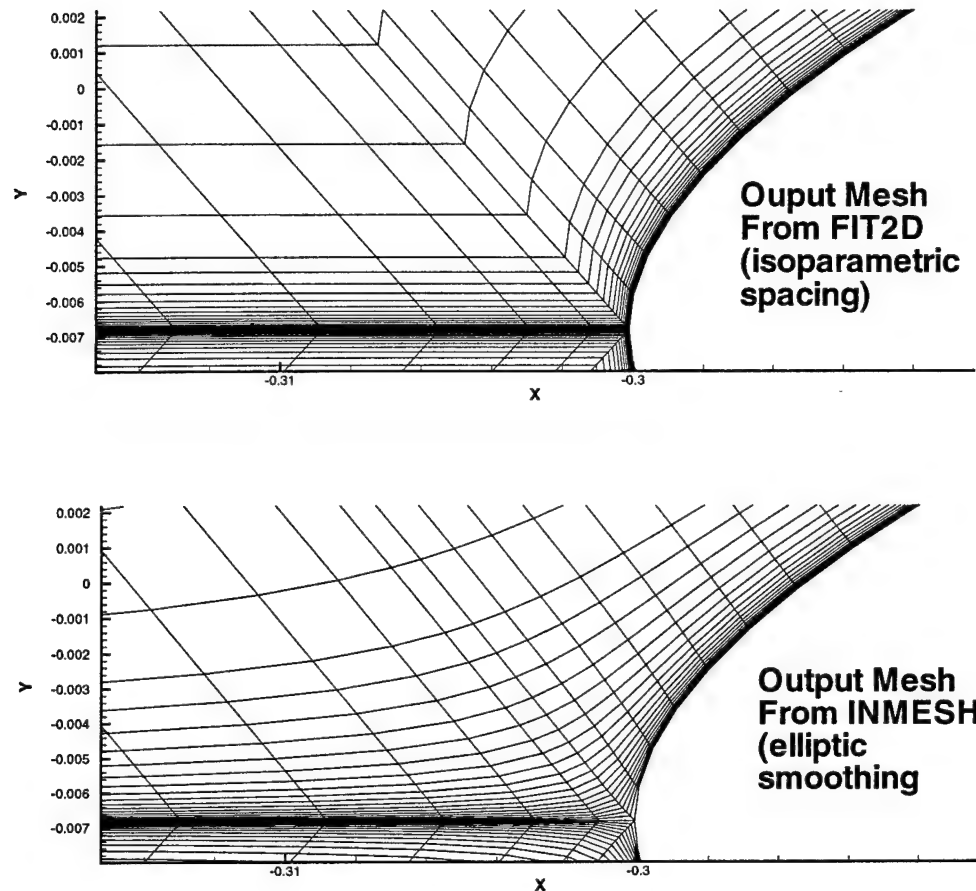


Figure 3-11: Typical Grid Spacing Before and After Smoothing with INMESH

3.6.2 Correction of INMESH output for use in DTNS2D

The INMESH code is used for a variety of grid generation applications. It provides the input files for the DTNS family of RANS solvers. INMESH writes out DTNS input files that overlap the individual zones by one cell at each adjoining boundary which are used if higher order boundary conditions are desired. The DTNS2D variant

of the code does not support this overlapping geometry configuration. Therefore a small computer code is used to condition the INMESH output so that it is compatible with the DTNS2D code. The program is called PATCH and it is listed in Appendix E. PATCH is specifically tailored for a six zone “H” grid scheme.

3.7 The RANS Solver

3.7.1 Overview of DTNS2D

The flow solver used in this thesis is the David Taylor Navier-Stokes Two Dimensional computer code developed by Gorski[13]. The DTNS2D computer code has been validated with experimental results for flow around two dimensional lifting bodies by Nguyen[25].

The DTNS2D code solves the RANS equations for incompressible turbulent flow which are derived in section 2.6. The flow domain is discretized into a large number of quadrilateral cells. The solution method is a cell centered finite difference method based on a finite volume derivation. The program is structured so that the flow domain can be broken into a number of separate blocks each having its own set of boundary conditions. This allows for modeling of flow around foils in tunnels or unbounded domains. Zone junctions support a number of possible boundary conditions such as: no-slip, tangent, continuity, pressure specified, or velocity specified. A typical zone scheme and associated boundary conditions was demonstrated in Figure 3-3.

The code utilizes a third-order upwind difference total-variation-diminishing (TVD) scheme applied to the convection terms and a second-order central difference scheme applied to the diffusion terms. A first order Runge-Kutta forward time marching scheme is used to determine the steady state solution.

Chapter 4

Data Post Processing Methods

4.1 Introduction

A variety of tools were used for post processing of the RANS output data. At a minimum, it was desired to obtain the lift and drag coefficients for a foil at a given Reynolds number and angle of attack. For the purposes of comparing the RANS solution with other foil analysis computer codes, the pressure distribution on the foil surface was also required. During the study, other questions arose regarding specifics of the flow characteristics within the boundary layer on the foil surface and in the wake. For studying foils in tunnels, it is beneficial to be able to extract tunnel wall boundary layer profiles.

4.2 UNNS2D

The output from DTNS2D contains all the flow characteristics at the center of each cell. However, the data is not in a readily viewable or workable format. A data conversion program UNNS2D, written by Scott Black of the MIT-MHL, was used to convert the data from *raw* output to a format compatible with TECPLOT v7.0. UNNS2D is a simple program and it could be easily adapted to convert data to any popular graphics software data format. The FORTRAN 77 source code for UNNS2D is located on the MIT-MHL software archive server. Once the data is loaded into TECPLOT, it can be displayed and manipulated in a variety of ways to view field

pressure and velocity contours or to draw streamlines. The UNNS2D output file is called *dtns2d.dat*. It contains data in columns by zone for each cell centered coordinate in the following order: x , y , u , v and P .

4.3 Computing Lift and Drag

Lift and drag are computed using the program FLD2D (Foil Lift and Drag Two-Dimensional). It is an adaptation of a program that was previously used in the MIT-MHL to calculate the lift and drag for the duct and body components on ducted axisymmetric bodies. FLD2D extracts the cell centered flow properties from the DTNS2D output files. The lift and drag are numerically integrated using the methods of section 2.3.3. The primary outputs from FLD2D are the shear component of the drag coefficient and the drag and lift coefficient obtained by integration of the surface normal pressure. The shear component of the lift is neglected because it is a very small quantity when compared to the lift obtained by pressure. A plot file called *cp.dat* contains data for the pressure distribution (C_P) on the foil and numerical trail for the integrated quantities. The FORTRAN 77 source code for FLD2D is also located on the MIT-MHL software archive server.

4.4 Bounding Box Calculations

One focal point of this thesis is to evaluate differences between computational results and experimental measurements. In the MIT-MHL Water Tunnel the CONTOUR bounding box program is used to calculate the lift and drag of 2-D foils from measured velocities around a bounding contour. The contour integration techniques used by the program CONTOUR are presented in section 2.5.1. A bounding box can be extracted from the *dtns2d.dat* file using TECPLOT and subsequently processed by the CONTOUR program. This is useful for a variety of comparisons. For instance, a side by side comparison of contour normal velocities can be made between identical

boxes used in an experiment and an equivalent geometry RANS solution. Bad data points in experimental measurements can be identified. Or if good experimental data is available, poor modeling in the RANS code or grid scheme can be identified. Additionally, bounding box calculations when applied to the RANS output data, provide a validation of the surface pressure integration routine(FLD2D) for lift and drag.

4.5 Extracting Velocity Profiles in the Boundary Layer

In the early stages of this thesis, research focused on the validation of previously obtained lift and drag characteristics of foils. It became apparent that the details of the flow inside the boundary layer near the foil and tunnel walls were also of interest. Using the LDV apparatus, it is possible to take velocity data at a sufficient number of points inside the boundary layer to get an accurate representation of the profile. The same data can be extracted easily from the RANS output using the TECPLOT data extraction feature. Once the profiles are extracted, characteristics of the boundary layers can be quantified using the methods of section 2.7.

4.5.1 Post Calculation of y^+

A validation check of the grid adequacy in the boundary layer region is obtained by extracting RANS velocity and position data from the first several cells above the foil surface. A reverse calculation of the y^+ value at these three locations is performed. The y^+ values for cell spacing are compared with the guidance in section 2.7.4. The computer code YPLUS is the small program used for this task. The FORTRAN 77 source code for YPLUS is located on the MIT-MHL software archive server.

The results for lift and drag are highly dependent upon the spacing of the first several cells above a no-slip surface. To demonstrate this a foil was analyzed using the FIT2D/DTNS2D analysis tool. For each case, the spacing of the first several cells above the foil surface was varied. The maximum y^+ used for the first cell was

approximately twenty. The minimum y^+ value used was less than one. All other grid parameters were held constant. Figure 4-1 shows plots of the pressure coefficient for each case. The cases corresponding to larger initial y^+ values yield higher lift

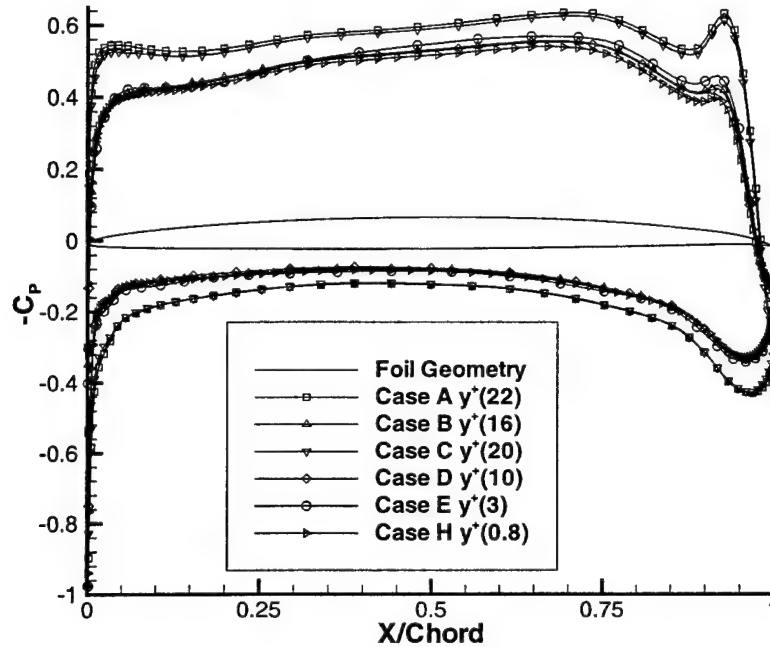


Figure 4-1: Pressure Distribution Dependence on y^+ Parameter

coefficients. The pressure distribution for the final case, with y^+ set at 0.8, compares well with other numerical results.

4.6 Extracting Velocity Profiles in the Wake

It was suspected that the turbulence models used in DTNS2D were inadequate. Data were extracted from the RANS solution in vertical cuts downstream of the trailing edge of the foil. These data were compared to identical wake cuts measured experimentally. Once again, TECPLOT was useful for extracting this data. These results are presented in Case Study I for the HRA foil.

Chapter 5

Results

In this chapter, two case studies are presented. The first is an analysis of a proposed advanced foil section at several angles of attack and a single Reynolds number. The flow around the foil is computed using DTNS2D in unbounded and bounded flow domains. Unbounded results are compared with the 2-D lifting surface analysis code PAN2D-BL. Bounded results are compared with experimental measurements. Differences and similarities are highlighted. A simple correction scheme is presented which extrapolates bounded measurements for lift to unbounded values.

In the second case study, a foil with a highly cambered trailing edge is analyzed at a single angle of attack and Reynolds number. The foil is analyzed using DTNS2D in unbounded and bounded flow domains. Unbounded and bounded results are compared with other computer codes and experimental measurements. This foil has proven to be very difficult to analyze with current numerical methods. The results of various computer codes are different enough that further detailed experimental study is warranted. The results in this case study are used to indicate regions of the flow around the foil where additional experimental data should be obtained. This case study demonstrates how RANS analysis can be useful for developing a streamlined experimental test plan which will maximize the useful data obtainable while minimizing the costly resources and time associated with experimental study.

These case study results are based upon DTNS2D solutions obtained from nu-

merical grids that demonstrated convergence with optimum y^+ spacing. The specific parameters used for the generation of these grids are presented in Appendix B.

5.1 Validity of the Results

5.1.1 Sources of Error

There are several potential sources of error in the DTNS2D RANS results. There is no provision in the computer code for modeling the laminar regime of a boundary layer. DTNS2D uses a boundary layer model that starts out fully turbulent from the leading edge of the foil or the beginning of a wall. Along the walls this error is probably not significant. On the foil, this effect may or may not be significant. The real transition point from laminar to turbulent flow is different on the pressure and suction sides of the foil. The errors resulting from the omission of modeling the laminar boundary layer and transition are difficult to quantify. Intuitively, the magnitude of this error is most likely dependent upon the Reynolds number, the camber and thickness distributions and upon the angle of attack.

Another phenomenon which may occur is that the turbulent boundary layer may revert back to a laminar flow after transition has occurred in certain regimes of Reynolds number. This effect is nearly impossible to quantify due to the stochastic nature of the process. But, nonetheless, it is another aspect of the real fluid flow that is omitted in the RANS solution which may have some impact on the overall results.

5.1.2 General Comments Regarding Comparison with Tunnel Experiments

In experimental tests, boundary layer transition on the foil is usually forced to occur on the foil at a specific location by the introduction of a turbulent stimulating surface irregularity such as bumps, rivets or a trip wire. At the location of the turbulence stimulators turbulent transition will occur with a fair degree of certainty. Forcing transition allows a more equitable comparison between experimental measurements

computer codes such as PAN2D-BL and XFOIL. By tripping the real flow in the experiment and specifying this same location in the computer code, one can consider the flows to be identical.

Although the foil section mounted in a tunnel is two-dimensional, the flow within the tunnel still retains some three-dimensional effects. In a pure two-dimensional numerical flow solver, the side walls are completely omitted. In the real tunnel, the side walls tend to have uneven boundary layer growth on the areas above and below the foil. The magnitude of these effects are just now receiving attention and are being evaluated through related research in the MIT-MHL[19].

The RANS computer code models absolute steady state conditions that are impossible to achieve in a water tunnel. In the tunnel there are minor variations in inflow velocity. In the real flow unsteady vortex shedding can occur at the trailing edge. Very thin foils operating at high lift coefficients may twist and flex under the loading. The tunnel is built to standard engineering tolerances. The measured angle of attack in the tunnel can only be set to within several hundredths of a degree. While in the gridded model the geometry is precisely specified. All these issues combine and add to the resulting uncertainty of the experimental results.

5.2 Case Study I: The HRA Foil

A proposed advanced foil design created by Hydrodynamics Research Associates(HRA) was selected for this case study. A significant amount of experimental data has been recorded for this foil in various configurations in the MIT-MHL Water Tunnel test facility. Therefore, it is a natural candidate for use in validation of the FIT2D/DTNS2D analysis tool.

The HRA foil section, presented in Figure 5-1, is intended for application in propeller blade design. This foil is innovative because of its reduced camber distribution near the leading edge for shock free entry. The moderate camber aft allows the foil

to sustain a broad uniform pressure distribution on the suction side enabling it to operate at a substantial lift coefficient while minimizing the potential for cavitation inception. The foil offsets are contained in Appendix F.1.

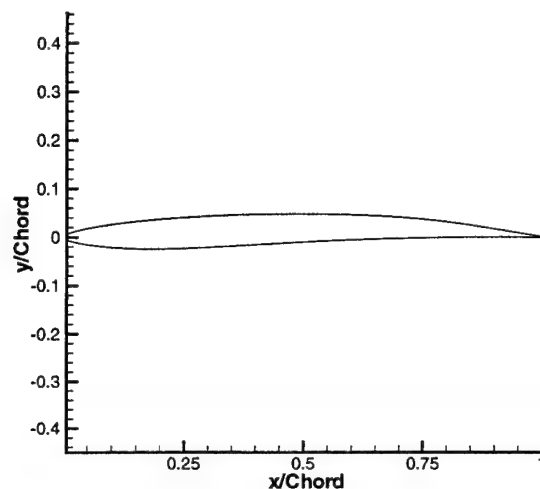


Figure 5-1: The HRA Foil Shape

5.2.1 Validation Check of Grid Adequacy

As a first check of the validity of the results obtained from DTNS2D, the y^+ grid spacing and the boundary layer profiles were analyzed. Within each set of bounded and unbounded runs, the only parameter varied was the angle of attack. Therefore, only two different FIT2D control files were necessary: one for the bounded runs and one for the unbounded runs. The grids were identical except for the small variations associated with changing the angle of attack. Table 5.1 summarizes the post calculation of the y^+ parameter for the unbounded and bounded runs. The results shown are for the runs conducted at $AOA = -0.28^\circ$. They are representative of all angles of attack that were analyzed. The first cell spacing falls within the target goal and those remaining conform to the distribution requirements specified in Section 2.7.4.

The next check is to ensure that the boundary layer conforms to the *Law of the*

Table 5.1: Post Processing Check of y^+ for HRA Foil Grid

	y^+ Value	y^+ Value
Cell Number	Unbounded	Bounded
Target for 1 st	1.0	1.0
1 (actual)	0.6	0.8
2	2.0	2.0
3	3.6	3.8

Wall. The boundary layer profiles shown in Figure 5-2 are extracted at the mid chord of the suction side of the foil. These profiles are consistent over a large portion of the foil surface except near the leading and trailing edges. At the leading and trailing edges there is a significant amount of curvature and fairly steep pressure gradients. It is natural to expect deviation from the Spalding formula because it is based on flat plate theory. Based upon the y^+ spacing and B-L profile results, the first indication

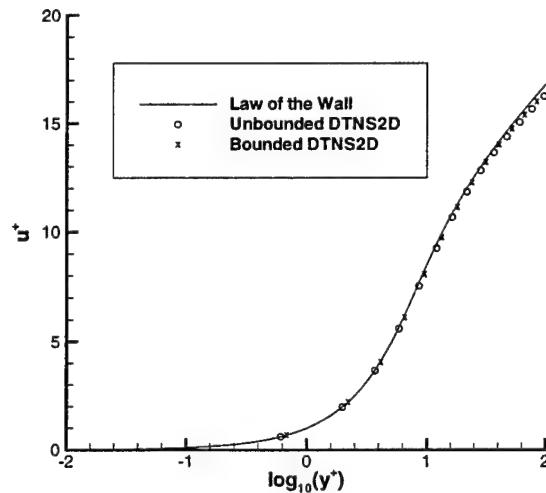


Figure 5-2: Comparison of DTNS2D Computed B-L Profile with Spalding Formula for HRA Foil

is that the results should be acceptable. The next step is to compare these solutions with other analysis methods.

5.2.2 Unbounded Flow Comparison

In the unbounded case the lift and drag coefficients obtained from the RANS calculations are compared with PAN2D-BL. A typical comparison of the foil surface pressure distribution is presented in Figure 5-3. This plot is prepared for a single case of the HRA foil at a 1° angle of attack. Comparisons at the other angles of attack evaluated are similar. Throughout a range from $-2^\circ \leq \alpha \leq +1^\circ$ the FIT2D/DTNS2D analysis tool agrees with PAN2D-BL to within one percent of the lift coefficient. FIT2D/DTNS2D slightly over-predicts the total lift which is evident from the difference in the pressure distribution near the trailing edge as shown in the inset panel of Figure 5-3. Values for drag coefficient computed by RANS and PAN2D-BL are comparable. Drag calculated from the RANS data is about 4% higher than the PAN2D-BL results throughout the angles of attack studied. A summary plot of these results is also presented later in Figures 5-6 and 5-7.

5.2.3 Bounded Flow Comparison

For the bounded case the lift and drag coefficients obtained from the RANS calculations are compared with experimental results. Throughout a range from $-2^\circ \leq \alpha \leq +1^\circ$ the FIT2D/DTNS2D analysis tool agrees with the experimental results to within a few percent of the lift coefficient. FIT2D under-predicts lift compared to the experimental results. Some error may be caused by position measurement error in the water tunnel experiment.

Values for drag coefficient computed by RANS and from experimental measurements are not as consistent as they are for the unbounded case. Drag calculated from the RANS data does not follow the same trend as the experimental measurements. At lower lift coefficients, such as $AOA \leq -0.5^\circ$, RANS results agree well with the experimental results. There is a large jump in the RANS computed drag for higher angles of attack. This is inconsistent when compared with the trends of the unbounded

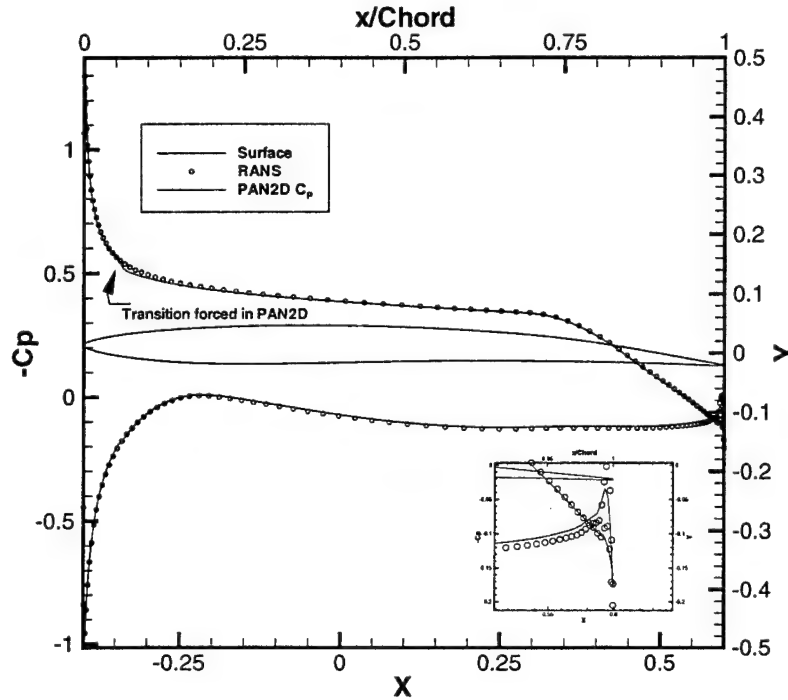


Figure 5-3: Comparison of Pressure Distribution on HRA Foil Using DTNS2D and PAN2D-BL($Re = 3 \times 10^6$, $AOA = 1^\circ$)

results and calculations. This may be caused by inadequate discretization of the flow domain or from a breakdown of the turbulence model under these flow conditions. A summary plot of these results is also presented later in Figures 5-6 and 5-7.

5.2.4 Bounding Box Comparison

Bounding box velocity contours extracted from the RANS data are used as input for the CONTOUR program. The results from the contour integration of the RANS data along with the surface integration results are presented in Table 5.2. The contour integration results compare favorably with the surface pressure integration results for all of the unbounded data. This indicates that the RANS solution is numerically consistent throughout the domain. In the bounded cases, however, the CONTOUR and surface pressure methods do not agree. It is not necessarily clear which method is in error for the bounded cases.

Table 5.2: CONTOUR and FLD2D Results for Lift and Drag Characteristics of the HRA Foil

AOA	Lift Coeff		Drag Coeff	
Unbounded	FLD2D	CONTOUR	FLD2D	CONTOUR
-2.00	0.0171	0.0173	0.0083	0.0098
-0.636	0.1623	0.1617	0.0083	0.0090
-0.280	0.2004	0.1995	0.0082	0.0090
0.885	0.3239	0.3224	0.0085	0.0092
Bounded	FLD2D	CONTOUR	FLD2D	CONTOUR
-2.00	-0.0144	0.0168	0.0091	0.0090
-0.636	0.1628	0.2113	0.0089	0.0111
-0.280	0.2107	0.2630	0.0103	0.0103
0.885	0.3620	0.3869	0.0102	0.0087

A sample contour obtained from the DTNS2D results and an MIT-MHL experiment are shown in Figure 5-4. The $\mathbf{V} \cdot \mathbf{n}$ component of the fluid velocity around the contours are plotted for each leg of the contour. The contour normal velocities are nearly identical for the upstream, downstream and top contour legs. The bottom contour leg data does not agree aft of the mid chord of the foil.

5.2.5 Wake Profile Comparison

To make an observation of the wake characteristics aft of the foil, vertical cuts of data were taken at successive locations downstream of the trailing edge. This data was taken at the same locations both experimentally and from the computational results. These cuts are presented in Figure 5-5. The first profile is taken 0.04 chord lengths downstream of the trailing edge. The experimental data and the RANS data are nearly coincident at this location. This implies that at points very close to the trailing edge the RANS solver is accurately capturing the true flow conditions in the vicinity of the trailing edge. The successive cuts still show good agreement. However, it can be observed that the RANS calculated wake is diffusing at a faster rate than the experimental wake. This may be due to ineffective turbulence model application by the RANS solver in the wake region.

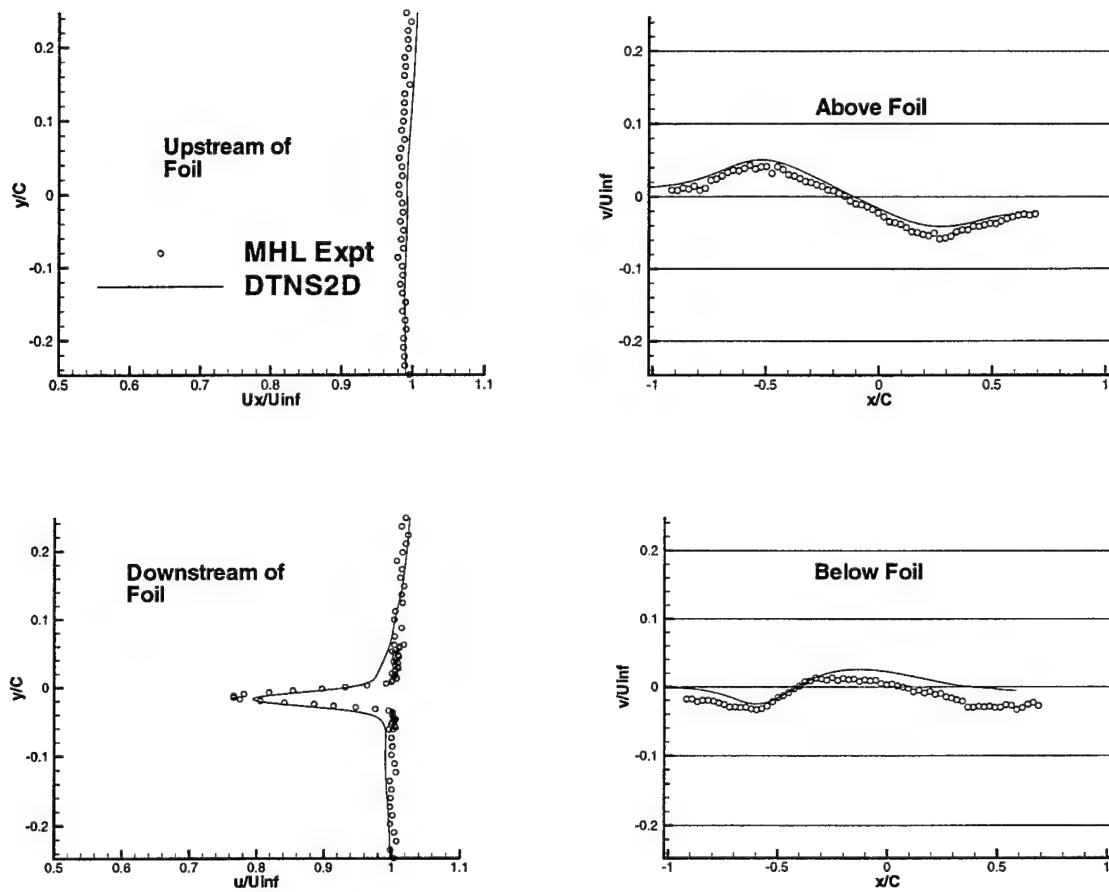


Figure 5-4: Comparison of Normal Velocity Component Around Contour Box for DTNS2D and HRA Experimental Results($Re = 3 \times 10^6$, $AOA = -0.636^\circ$)

5.2.6 Relating Bounded Measurements to Unbounded Characteristics

The unbounded and bounded results for lift are summarized in Figure 5-6. It is desirable to develop a simple scheme to relate measurements taken in the tunnel to their unbounded values in the absence of walls. There are several options for implementing such a scheme. The key decision regarding which type of scheme to use depends on whether you believe the experimental results or the numerical results to be more accurate. Rather than debate the merits and faults of these results, it is assumed here, for illustrative purposes, that the experimental results are accurate and have been obtained with a high level of confidence.

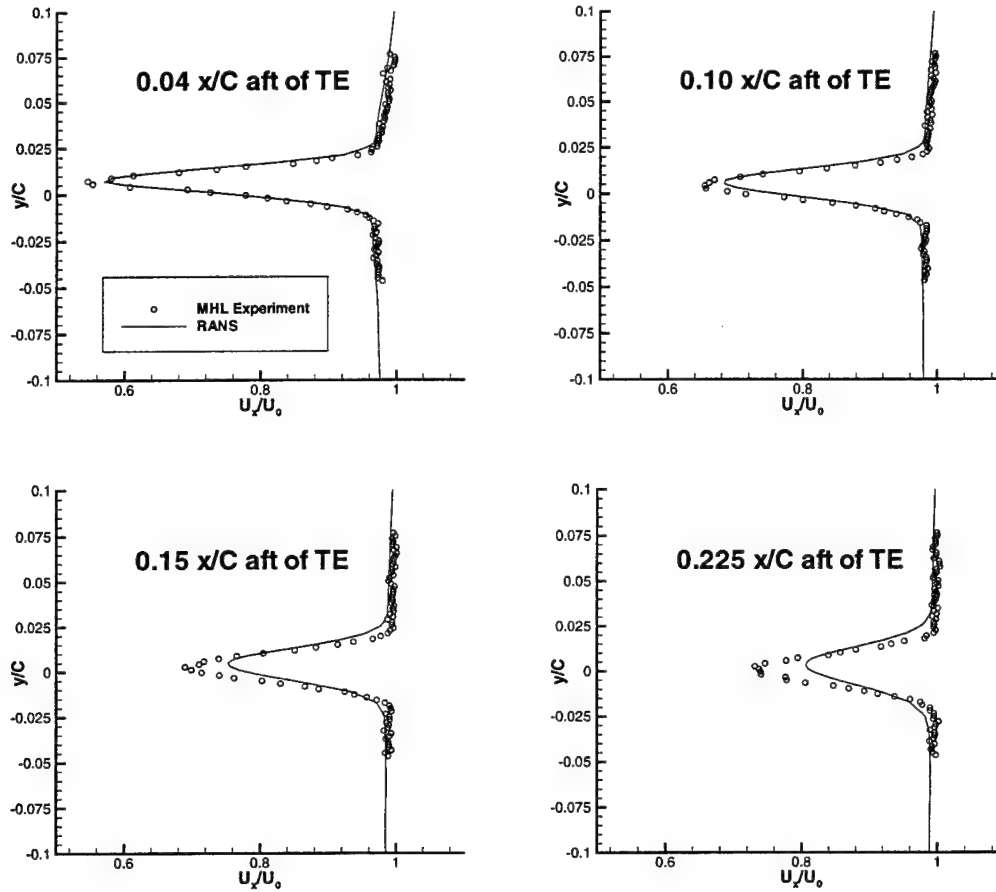


Figure 5-5: Comparison of Wake Profiles for DTNS2D and HRA Experimental Results($Re = 3 \times 10^6$, $AOA = -0.28^\circ$)

Given this assumption, a method is required to correct the results obtained from the unbounded computer code estimates to unbounded *true* results. The method formulated here neglects side wall boundary layer effects. It is assumed the RANS solver is fairly accurate at capturing the magnitude of the relative effects between the bounded and unbounded cases. Referring back to Figure 5-6, each of the lift curves is linear in the regime studied. The unbounded results for DTNS2D and PAN2D-BL are close enough that the results can be considered equal. A least squares linear regression for the lift curves yields the results for lift slope and intercept presented in Table 5.3. Each lift curve can be represented with the following equation:

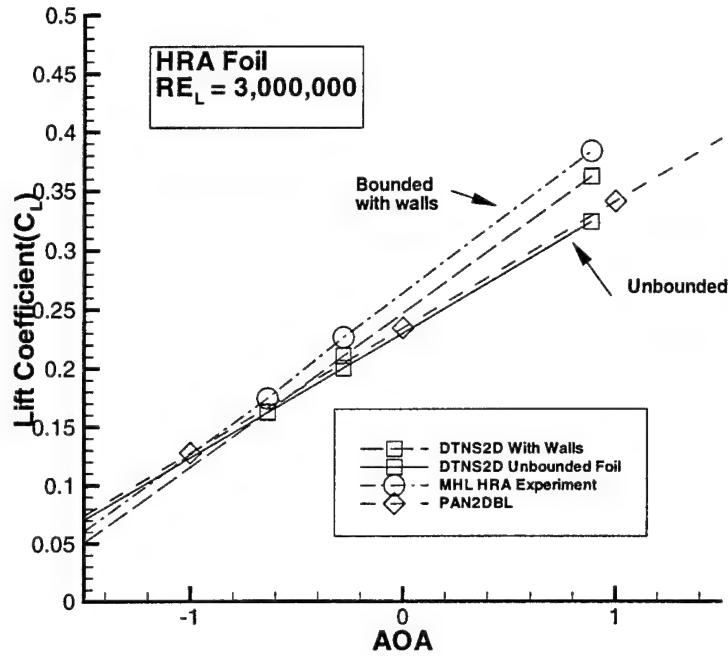


Figure 5-6: Summary of Results for Lift Coefficient vs. AOA for the HRA Foil

Table 5.3: Lift Slope and Intercept Data for HRA Foil Calculations

Method	Slope(a)	y -axis Intercept(b)
MHL Expt.	$0.137 C_L/AOA$	$0.267 C_L@AOA = 0^\circ$
DTNS2D Bounded	0.131	0.247
PAN2D-BL/DTNS2D(unb)	0.107	0.234

$$C_L = a \times AOA + b, \quad (5.1)$$

where a and b are the respective slopes and intercepts. The difference in the slopes and intercepts of the bounded DTNS2D calculated curve and the experimentally measured curve represent the relative error between the estimated and actual result. Slope and intercept correction factors are formulated by:

$$\begin{aligned} a_{cf} &= a_{expt} - a_{DTNS2D(bounded)} \\ b_{cf} &= b_{expt} - b_{DTNS2D(bounded)} \end{aligned} \quad (5.2)$$

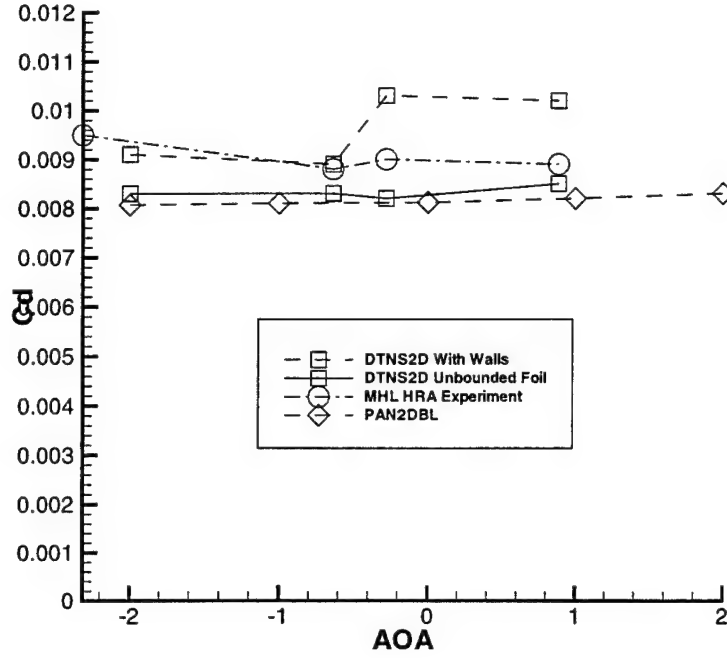


Figure 5-7: Summary of Results for Drag Coefficient vs. AOA for the HRA Foil

Once side wall boundary layer effects have been quantified, additional terms could be placed on the right hand side in equation 5.2. The correction factors are applied to the unbounded numerically derived lift curve in the following manner:

$$(C_L)_{cor} = (a_{unb} + a_{cf}) \times AOA + (b_{unb} + b_{cf}). \quad (5.3)$$

Equations 5.2 and 5.3 are applied using the values presented in Table 5.3. Figure 5-8 shows the corrected lift curve compared with the previously obtained results.

What is learned from this case study is that a RANS solver can be used as a *liaison* between bounded and unbounded flows. The FIT2D/DTNS2D analysis tool agrees well with current state of the art unbounded foil analysis tools such as PAN2D-BL. In bounded flow the FIT2D/DTNS2D analysis tool agrees well with experimental measurements. Therefore, to conduct a complete analysis for a given lifting flow, where comparison with experimental results are desired, the following methodology

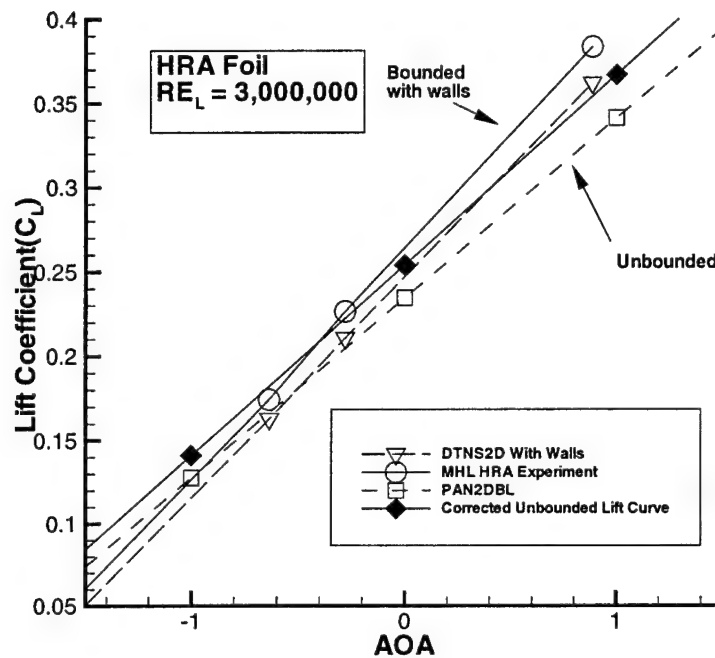


Figure 5-8: Lift Curve With Correction Factors Applied for HRA Foil

is proposed:

1. Based on the fact that FIT2D/DTNS2D and PAN2D-BL agree well for unbounded flow, use PAN2D-BL to evaluate the foil in the range of angle of attack and Reynolds numbers that will be used in the experimental study.
2. Select a few angles of attack and Reynolds numbers to evaluate using the FIT2D/DTNS2D tool for the bounded case.
3. Obtain experimental results and compare the results to the bounded FIT2D/DTNS2D results.
4. Using equations 5.2 and 5.3, compute the correction factors for the unbounded predictions and calculate the corrected lift curve.

The above methodology should make efficient use of computational assets and result in a consistent comparison method between experimental and numerical results. The corrected lift curve provides an estimate of the error between the actual measured lift and drag of a foil and the numerically predicted unbounded lift and drag. The long term goal of the correction factor scheme is to provide motivation to improve

measurement and numerical prediction techniques until the correction factors are reduced to zero. Once the correction factors are reduced to zero then it can be concluded that the numerical computer codes have captured all the essential physics of the problem.

5.3 Case Study II: Foil With A Cupped Trailing Edge

The development of the “Cupped Foil” is interesting. The parent shape of the cupped foil is derived from a NACA 0016 thickness distribution foil with an $a = 0.8$ mean line and a maximum camber of 2.55%. The parent foil was given the name “B1” and is representative of a typical destroyer propeller blade section at $r/R = 0.7$. The trailing edge was beveled using the standard U.S. Navy formula for anti-singing trailing edges.

Bloch suggested that it was possible to design a foil with a substantially higher lift to drag ratio by shifting the maximum camber distribution aft and *cupping* the trailing edge downwards[7]. This conclusion was based on a series of numerical results from the computer code XFOIL. Subsequently, Jorde[16] conducted experiments to compare the B1 parent foil to the B1 modified with a cupped trailing edge. The modified B1 foil section with the cupped trailing edge is presented in Figure 5-9. The foil offsets are contained in Appendix F.2.

Although the cupping at the trailing edge is not extreme to the human eye, it is proving to be a substantial challenge to the various array of computer codes available for numerical analysis. Currently there is no agreement among the computer codes that have been used to analyze the foil. Therefore, it is an interesting foil to use for this case study as a validation of the FIT2D/DTNS analysis tool. The first goal in this case study is to identify differences between the FIT2D/DTNS2D tool and other computer methods. This information will be used to provide guidance for future experimental research involving this foil as to where it would be productive to gather

detailed data. Once a suitable database of experimental data is obtained reflecting the actual conditions of the flow around the foil, then changes can be incorporated in the computer codes to make their methods more accurate. The difficulties associated with the unique geometry of the cupped B1 foil provide a valuable opportunity to improve the robustness of state of the art computational tools.

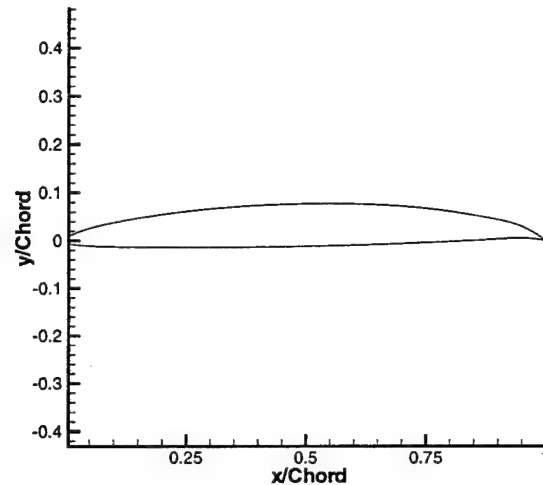


Figure 5-9: The B-1 Foil With Cupped Trailing Edge Modification

5.3.1 Validation Check of Grid Adequacy

As with the HRA foil, the first check of the validity of the results obtained from DTNS2D is to check grid spacing and boundary layer profiles. The cupped foil was studied at one angle of attack and Reynolds number for unbounded and bounded calculations. Two different control files were necessary: one for the bounded run and one for the unbounded run. Table 5.4 summarizes the post calculation of the y^+ parameter for the unbounded and bounded runs. The results shown are for the runs conducted at $AOA = 0.5^\circ$ and $Re = 3 \times 10^6$. The first cell spacing falls within the target goal and those remaining conform to the distribution requirements specified in section 2.7.4.

Table 5.4: Post Processing Check of y^+ for B-1 Cupped Foil Grid

	y^+ Value	
Cell Number	Unbounded	Bounded
Target for 1 st	1.0	1.0
1 (actual)	0.8	0.6
2	2.1	2.0
3	4.0	3.6

The next check is to ensure that the boundary layer conforms to the *Law of the Wall*. The boundary layer profiles shown in Figure 5-10 are extracted at the mid chord of the suction side of the foil. These profiles are consistent over a large portion of the foil surface except near the leading and trailing edges. At the leading and trailing edges there is a significant amount of curvature and fairly steep pressure gradients so it is natural to expect deviation from the Spalding formula which is based on flat plate theory.

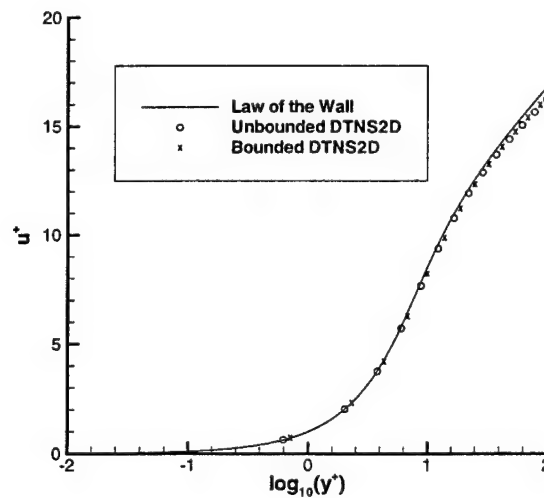


Figure 5-10: Comparison of DTNS2D Computed B-L Profile with Spalding Formula for B1 Cupped Foil

5.3.2 Unbounded Flow

Using the post processed DTNS2D output from the unbounded model, velocity and pressure contours can be viewed for the flow domain. Velocity contours near the trailing edge are presented in Figure 5-11. On this plot flow separation can be observed on the suction side of the trailing edge of the foil over the last three percent of the chord. This separation is de-cambering the flow at the trailing edge causing a significant reduction in lift coefficient from the inviscid value obtained from the VLM. The flow is also decelerating beneath the cupped region.

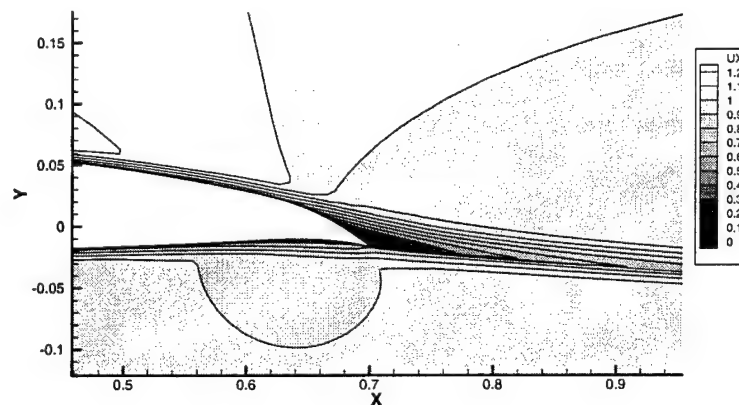


Figure 5-11: Velocity Contours Near Trailing Edge of Cupped Foil(unbounded flow)

Pressure contours around the leading edge of the foil are presented in Figure 5-12. The concentric circles of pressure contour point to the leading edge stagnation point. An interesting observation of the pressure contours is that the value of the stagnation point pressure exceeds the pressure that is analytically possible. This could result from artificial compressibility introduced into the numerics of the RANS solver(an additional equation of state) to run incompressible solutions. Another possibility is simply numerical noise. The cell sizes near the stagnation point are very small. The calculations for those cells may be near the numerical precision limit of the CPU. This stagnation pressure error is observable in the data from all three of the RANS

solvers compared in this case study.

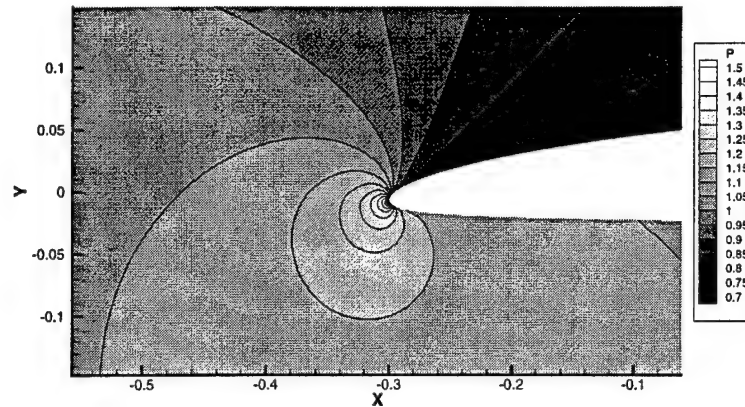


Figure 5-12: Pressure Contours Near Leading Edge of Cupped Foil(unbounded flow)

For the analysis of the flow around the cupped foil FIT2D/DTNS2D results are compared with a RANS study by C.I. Yang of DTMB and results obtained by Kerwin using PAN2D-BL. A comparison of the calculated surface pressure coefficient for the three methods is presented in Figure 5-13. In general, the three methods yield comparable results. The largest differences occur in the leading and trailing edge regions. The difficulty in comparing these results relates to the difference in leading edge stagnation pressure. PAN2D-BL has a maximum pressure coefficient(C_P) of exactly 1.0, the maximum in the RANS solutions are approximately 1.1 for DTNS and 1.15 for C.I. Yang. Correcting for these differences may yield better alignment of the pressure coefficient curves. In any event, the differences in the trends at the leading and trailing edges are significant enough to warrant detailed experimental study in these regions in order to validate the predictions of the different computer codes. Comparison of lift and drag coefficient results are presented in Table 5.5. The inviscid lift coefficient obtained by the VLM is double the viscous value obtained by RANS and PAN2D-BL. The viscous effects on lift are significant with this foil.

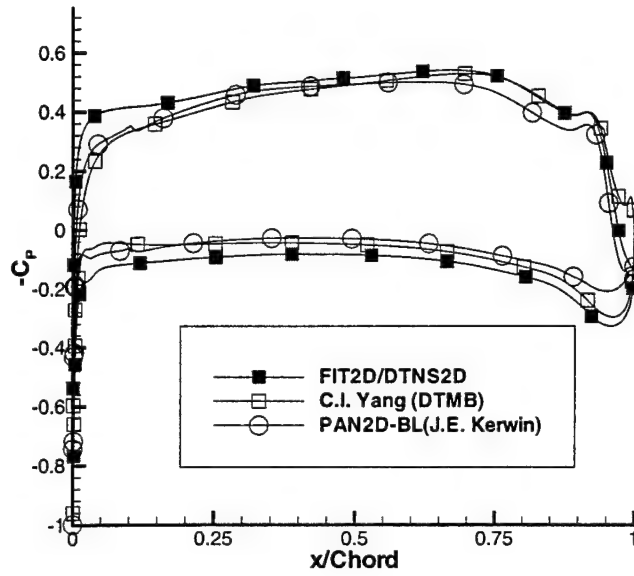


Figure 5-13: Comparison of Pressure Distribution on Cupped Foil Obtained by Two Different RANS Solvers and PAN2D-BL

Table 5.5: Comparison of Unbounded C_L & C_D Calculations for B1 Cupped Foil ($Re = 3 \times 10^6$, $AOA = 0.5^\circ$)

Method	C_L	C_D
VLM	1.0555	n/a
FIT2D/DTNS2D	0.579	0.0102
C.I. Yang RANS	0.505	–
PAN2D-BL	0.472	–
XFOIL	0.5281	–

5.3.3 Bounded Flow

As is demonstrated for the unbounded flow, velocity contours near the trailing edge are presented in Figure 5-14. On this plot flow separation is again observed on the suction side of the trailing edge of the foil over the last three percent of the chord. In the bounded case, the separated flow region is wider and extends farther aft than in the unbounded case. This separation is de-cambering the flow at the trailing edge causing a significant reduction in lift coefficient from the value obtained from the VLM. As with the unbounded case, the flow is decelerating beneath the cupped

region.

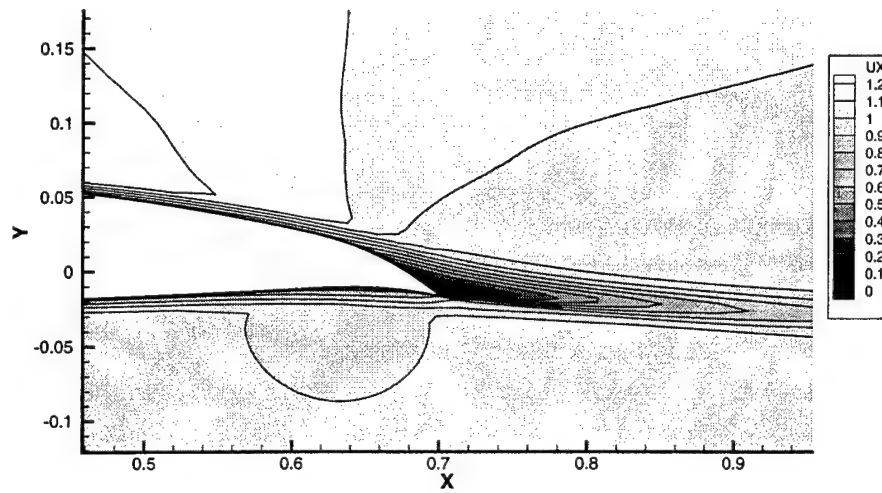


Figure 5-14: Velocity Contours Near Trailing Edge of Cupped Foil(bounded flow)

Pressure contours around the leading edge of the foil are presented in Figure 5-15. In similar fashion to the unbounded case the concentric circles of pressure contour point to the leading edge stagnation point. An anomaly in the figure is the discontinuity of one of the pressure contour lines forward of the leading edge. Close inspection of the data file indicates the anomaly is caused by a TECPLOT interpolation error across the zonal boundary. Additionally, as is observed in the unbounded case, the value of the stagnation point pressure exceeds the pressure that is analytically possible.

For the analysis of the flow around the bounded cupped foil, FIT2D/DTNS2D results are compared with two different RANS studies by C.I. Yang of DTMB and Lafe Taylor of Mississippi State University. A comparison of the calculated surface pressure coefficient for the three methods is presented in Figure 5-16. For the bounded case, the FIT2D/DTNS2D results are in closer agreement with the results by Yang than in the unbounded case.

Comparison of lift and drag coefficient results are presented in Table 5.6. There is

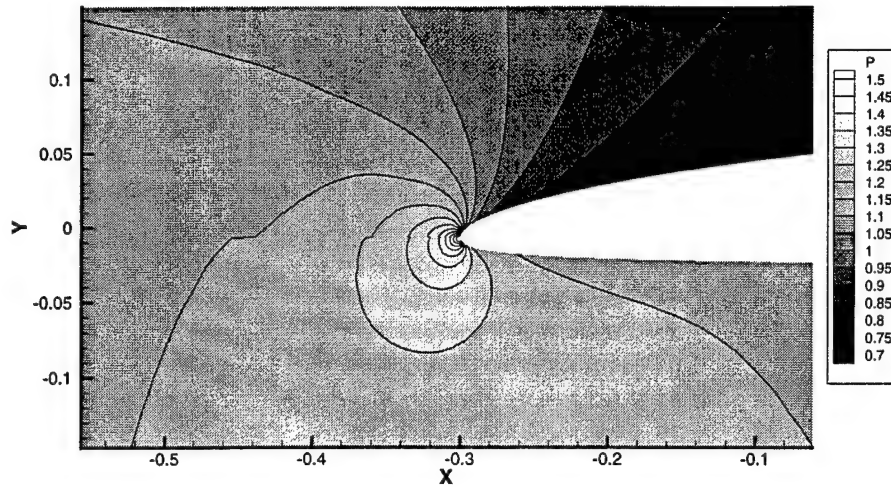


Figure 5-15: Pressure Contours Near Leading Edge of Cupped Foil(bounded flow)

Table 5.6: Comparison of Bounded C_L & C_D Calculations for B1 Cupped Foil ($Re = 3 \times 10^6$, $AOA = 0.5^\circ$)

Method	C_L	C_D
VLM	1.2017	n/a
FIT2D/DTNS2D	0.6413	0.0117
C.I. Yang RANS	0.605	—
Lafe Taylor RANS	0.578	—

a similar disparity between the inviscid results and the viscous results as is observed in the unbounded case. Once again, the viscous effects on lift are significant with this foil.

As one would expect, the presence of the tunnel walls in close proximity to the foil affects the lift and drag characteristics of the foil. Figure 5-17 demonstrates the difference in the resulting pressure distribution on the foil. To keep the comparison simple, only the FIT2D/DTNS2D results are shown. The tunnel walls cause a *flow cambering* effect to the fluid flow which increases the lift coefficient from the unbounded value.

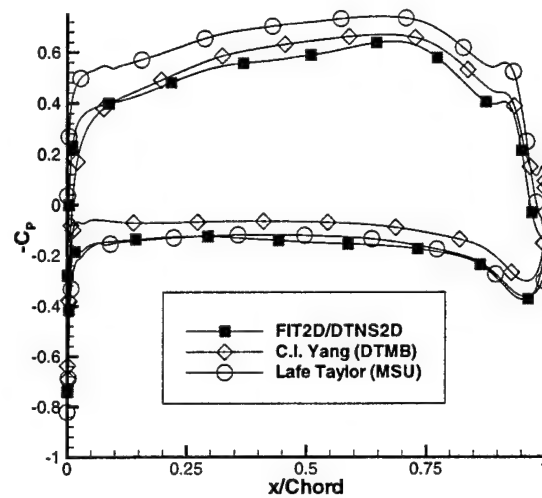


Figure 5-16: Comparison of Pressure Distribution on Cupped Foil Obtained by Three Different RANS Solvers

5.3.4 Tunnel Wall Boundary Layer Comparison

In the bounded case, the presence of the B1 cupped foil alters the boundary layer growth on the upper and lower walls of the tunnel. This effect is greater on the upper wall. A plot of the velocity contours throughout the domain is presented in Figure 5-18. On the upper wall, ahead of the foil, the boundary layer growth follows the typical trend for a flat plate. Above the foil the boundary layer thins. Aft of the foil the boundary layer grows rapidly. These changes in the boundary layer were initially observed in the RANS calculations and subsequently validated by MIT-MHL water tunnel LDV measurements. Boundary layer profiles for the upper tunnel wall are presented in Figure 5-19. The DTNS2D solution shows excellent agreement for the boundary layer characteristics ahead of the foil. Agreement remains good above the foil. In the region above and just aft of the trailing edge, the DTNS2D boundary layer profile deviates markedly from the experimental results. More detailed experimental measurements are required in this region to identify the exact location and conditions along the wall where the DTNS2D solution begins to deviate from the experimental

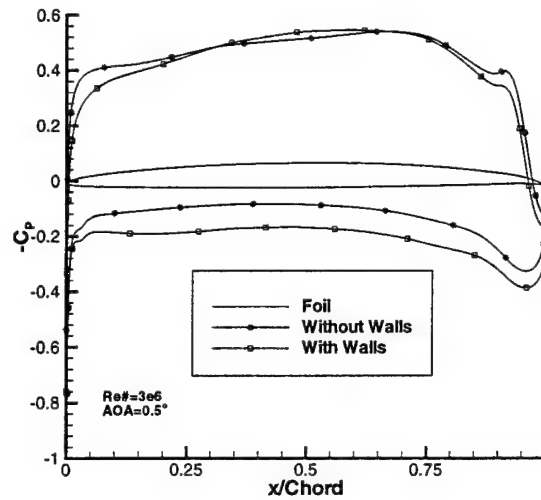


Figure 5-17: Comparison of Pressure Distribution on Cupped Foil for Bounded and Unbounded Flow

measurements.

Displacement thickness(δ^*) and momentum thickness (θ) integrals are computed at several locations along the upper wall. These results are presented in Figure 5-20. From the leading edge to the upstream limit of the flow domain, FIT2D/DTNS2D agrees with the experimental results. Downstream of the leading edge, the RANS results differ significantly from the experimental measurements. This may be a result of poor turbulence modeling in the regions of rapid pressure changes along the wall.

5.3.5 Separated Flow

Separation occurs near the trailing edge of the cupped foil at the angle of attack and Reynolds number used in this case study. The separated flow is clearly observable in Figures 5-11 and 5-14. Another interesting way to observe separated flow is through the use of streamlines. Streamlines are obtained in the TECPLOT software package by following velocity vectors in the domain from a specified starting point. Figure 5-21 shows streamlines obtained from the DTNS2D RANS solution. Separation occurs over the last 3% of the foil. Figure 5-22 obtained from the C.I. Yang results is

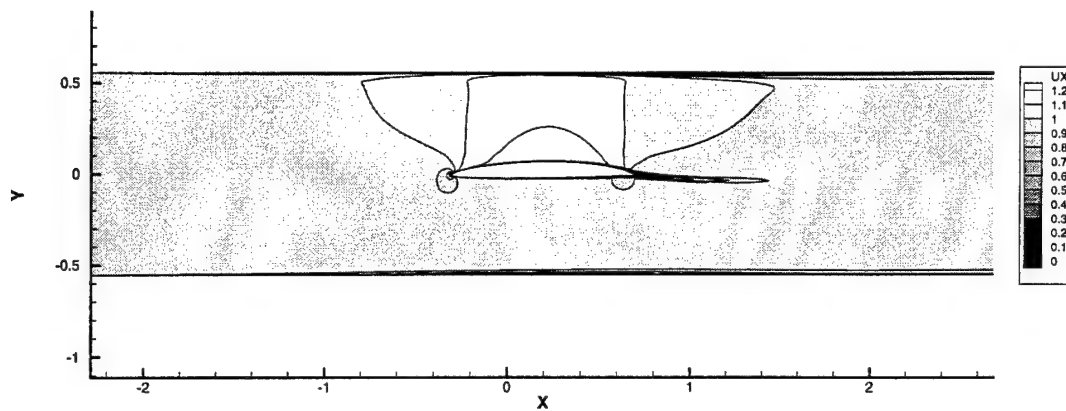


Figure 5-18: u Contours for Cupped Foil in Bounded Flow

included for comparative purposes. The separation zone in Yang's result is similar in length of the zone, but it is much wider across at the trailing edge. From the global perspective of the total flow, this difference is minimal but it is nonetheless present. The difference in the two RANS results indicates that detailed experimental data is required to validate the flow predictions at the trailing edge.

5.3.6 Developing An Experimental Test Plan

The cupped foil is a challenging foil to analyze numerically. There are many inconsistencies in the results among the different types of numerical solutions. More experimental data is required to validate the different aspects of the computer codes. Obtaining good experimental data is a very difficult task. Additionally, precious time and resources associated with experimental studies preclude obtaining data "everywhere". Here the RANS results can be applied to help identify the key areas that should be measured. Two benefits are realized. First, the possibility of taking data in areas of the flow that are not significant is reduced. Second, areas of interest that may be overlooked and not measured are reduced.

The cupped foil case study yields the following guidance for the experimentalist to conduct a study of the B1 Cupped Foil:

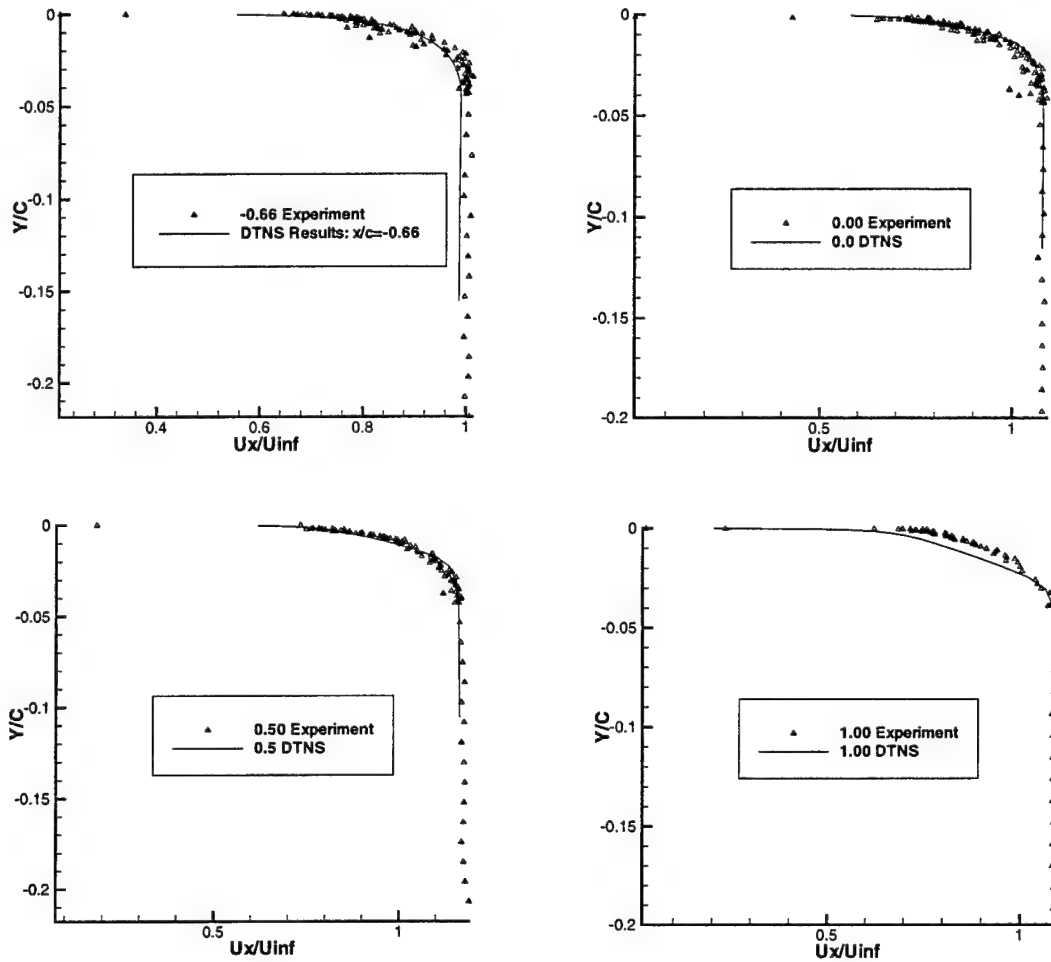


Figure 5-19: Upper Tunnel Wall Boundary-Layer Profiles

- The RANS solutions for the upper wall boundary-layer growth exhibit unusual characteristics. Boundary layer profiles should be obtained at several locations along the tunnel wall to confirm this. (see section 5.3.4)
- Separation occurs at the trailing edge of the foil. The extent of separation is not consistent among the RANS solvers. Detailed velocity profiles are required over the last 3% of the suction side of the foil with additional wake cuts down stream.
- The velocity contours on the pressure side of the foil near the “cup” are interesting. A moderate amount of data should be obtained in this region.

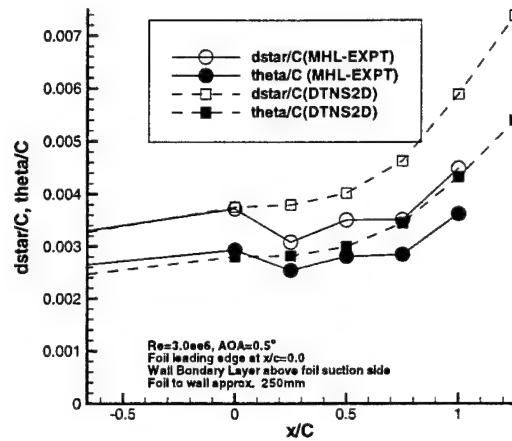


Figure 5-20: Upper Tunnel Wall Boundary-Layer Displacement Thickness and Momentum Thickness

- At the stagnation point, the pressure values obtained by the RANS solvers is non-physical. The stagnation point should be located experimentally and pressure measurements in the vicinity of the stagnation point should be obtained.
- The different numerical computer codes do not agree on the lift and drag characteristics of the foil. Lift and drag should be measured with statistical repeatability using bounding box contours. The lift and drag coefficients should be calculated and the $\mathbf{V} \cdot \mathbf{n}$ component on each leg of the contour should be compared with the identical contours in the RANS domain.

This provides the starting point for a detailed study. Based on the current numerical predictions, a lot of progress can be realized with the above test plan at a single angle of attack and Reynolds number. Once the above tasks are complete and the computer codes are validated, then the study can be expanded to a range of angles of attack and Reynolds numbers.

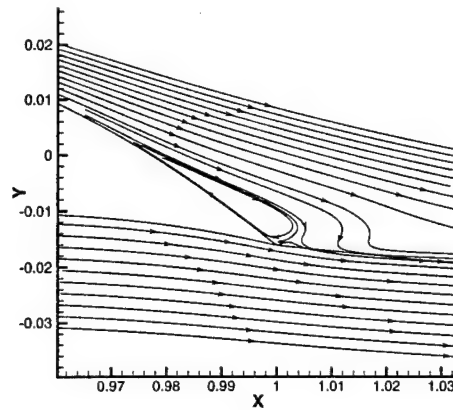


Figure 5-21: Streamlines Near Trailing Edge of Cupped Foil(DTNS2D Solution)

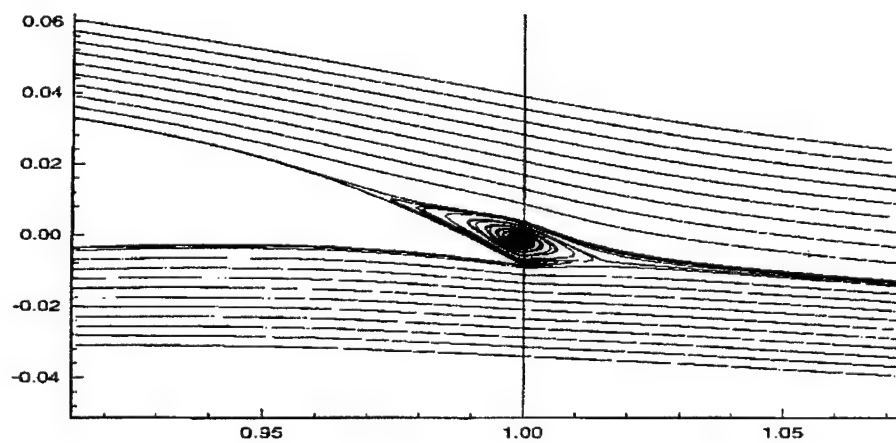


Figure 5-22: Streamlines Near Trailing Edge of the B1 Cupped Foil(C.I. Yang Solution)

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The computer code FIT2D has been developed to rapidly generate the geometry for the fluid flow domain surrounding an arbitrary foil shape at a specified angle of attack in the MIT Marine Hydrodynamics Laboratory (MHL) water tunnel. This geometry is provided as input data for the RANS solver DTNS2D. A suite of software tools have been developed to provide post processing analysis to compare the RANS solution with other numerical techniques and experimental measurements.

Through the use of the FIT2D/DTNS2D analysis tool, it has been shown that RANS solvers are quite useful for observing the details of the fluid flow around hydrofoils. The RANS results coupled with experimental data provide a good validation database for other numerical lifting analysis tools such as PAN2D-BL and XFOIL.

The importance of the y^+ parameter should be emphasized again. The cells in the grid near a no-slip surface must be sized to follow the guidance in section 2.7.4 in order to capture the significant viscous effects within the boundary layer. Failing to follow this guidance will undoubtedly yield questionable RANS solutions.

Adapting the fluid mesh zonal boundaries to the wake of the foil was somewhat successful. Alignment allowed better concentration of the finer discretization in the region of the steepest velocity changes. The wake adaption will offer significant advantages if the turbulence modeling in the wake is improved.

FIT2D is robust and it can be applied to a variety of foil geometries. With minor modification it could be used for multicomponent systems such as yacht sail and mast assemblies. A version of FIT2D has been converted for use to generate grids to represent axisymmetric ducted propulsor vehicles for use with the DTNS axisymmetric flow solver.

Our knowledge of the details of two-dimensional fluid flow around hydrofoils is incomplete. The current lifting analysis codes do not yet achieve the results that minimize the performance risk in proceeding from design to production of advance foil section propellers without extensive model testing. It is apparent from the two case studies presented in Chapter 5 that improvements in propeller design computer codes can be achieved by first looking back to the details of two dimensional lifting flows. When closure is brought to the remaining details of the 2-D lifting analysis codes, the lessons learned can be incorporated back into the propeller design codes. Then advanced foil sections will be able to be incorporated in future propellers without having to undergo the monumental effort that was required in the Advanced Technology Demonstration for propellers at the Carderock Division, Naval Surface Warfare Center.

6.2 Recommendations

The following areas require further research in order to advance the current level of knowledge of two dimensional fluid flow around lifting surfaces:

- Current turbulence models in computer codes do not work well in the wake region behind a foil. New models based on detailed experimental tests could do better at matching the rate of wake diffusion as the wake travels downstream. Improving the turbulence models will also improve the RANS solver's predictions in the boundary layer near a no-slip surface.

- Foils such as the B1 cupped foil are not easily modeled by current computer tools. More experimental study is required for these types of foils if their advantageous lift/drag characteristics are to be exploited in the future. The experimental results should be used to provide validation data for improved foil analysis computer codes.
- FIT2D should be modified so that it works better for generating grid geometry for unbounded flows. Current grid spacing algorithms cluster cells too finely at the outer regions of the flow domain.
- The DTNS2D RANS solver should be modified so that boundary layers start out laminar and then undergo transition to a turbulent boundary layer.
- The boundary layer growth on the side walls of a water tunnel is uneven. The magnitude of this effect is currently unknown. Detailed measurements are required to quantify this effect. Additionally, it would be interesting to compare these measurements with a RANS solution for an equivalent three-dimensional domain.
- The issue of excessive calculated stagnation pressure needs to be addressed in DTNS2D. It is not known if the problem results from artificial incompressibility within the code or if the code is not adequately imposing the specified pressure boundary condition at the downstream extent of the domain.

Appendix A

FIT2D Program Listing

```
PROGRAM FIT2D
C*****
C
C   Prepares all input files for program INMESH for 2D foils
C   operating in the MHL tunnel at various angles of attack.
C
C   Reads in foil geometry file, splines offsets
C   splits upper and lower surfaces adjusts to
C   angle of attack set in control file and prepares
C   input and restart files for INMESH.
C
C   2/97 JD
C   Incorporated ADAPT subroutine to use vortex lattice lifting
C   line to find wake dividing line to make grid wake adaptive.
C
C   3/97
C   Updated ADAPT subroutine to include image vortices due to walls
C
C   3/97
C   Updated input and adapt to accept grid line data for wake
C   from rans output.
C
C   WRITTEN BY: JOHN DANNECKER, MIT, 1996, Last Modified: 06 FEB 97
C*****
C   Program executive control file:
C
C   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
C   !
C   ! NOTE: A CARRIAGE RETURN MUST BE PLACED AT THE END !
C   !         OF ALL DATA FILES                          !
C   !
C   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
C
C   "fname.ctrl"
C       The ".ctrl" file contains the governing information
C       which specifies:
C           foil file: name of .naca or .foil file
C           AOA: foil angle of attack ref to free stream.
C           Pivot point: point about which foil is inclined
C           Scale Parameter: ratio of chord length to tunnel dim
C           USL: upstream flow domain limit in chord lengths
C           DSL: downstream flow domain limit in chord lengths
C           PHI1: leading edge zone offset number of chord lengths
C               rake zone forward at tunnel wall.
C           PHI1: trailing edge zone offset number of chord lengths
```



```

C
C File Format:      FILE HEADER (limit 40 Chars)
C                  1 (integer, this flags program to NACA type geometry)
C                  ## Chord Length
C                  ## Number of Stations
C                  Thickness(tmax), Camber(ymax)
C                  Station(x/C) 1/2 Thickness(t/tmax) Camber(y/Ymax)
C                  .
C                  .
C                  .
C                  Leading Edge Radius Multiplier

```

"fname.foil"

A "FOIL" geometry file contains the X,Y points that describe the surface of the foil. Foil data points shall be sorted so that the file marches from the trailing edge lower surface to the leading edge and then back to the trailing edge along the upper surface. If the two TRAILING edge points are NOT coincident, then fit2d.f will connect the two points with a straight line and specify an additional mesh zone extending from the blunt trailing edge downstream.

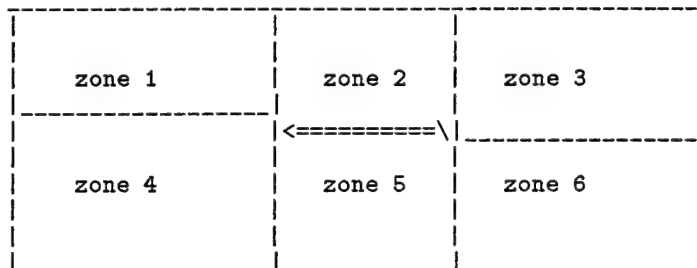
```

C File Format:      FILE HEADER (limit 40 Chars)
C                  2 (integer, this flags program to FOIL type geometry)
C                  ## Chord Length (for normalization)
C                  ## Number of Data Points
C                  X Y Training Edge Lower Surface
C                  .
C                  .
C                  X Y Leading Edge
C                  .
C                  .
C                  X Y Trailing Edge Upper Surface

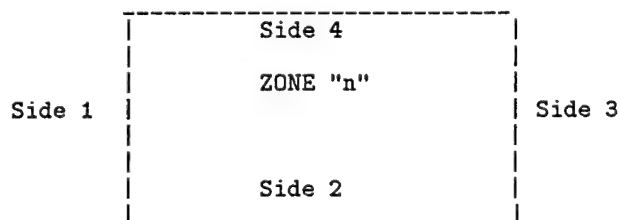
```

Here is the Zoning Scheme required by INMESH and DTNS2D:

Assemble the zones for input to transfinite interpolation.



Here is how an individual zone is specified:



C*****

```

IMPLICIT NONE
INTEGER NPOINTS, NFTYPE,MIDPRES,MIDSUCT,NOSE
INTEGER NUS, NDS, NVS, NTOP, NBOT, NTEMP,I,NI,NTEMP2,J,NTEMP3
INTEGER NSUCT1,NSUCT2,NPRES1,NPRES2,NPRESSURE,NSUCTION
INTEGER NUMIT,NRESTART,NWHAT,NLINE,NZONE,M,N
INTEGER NTOPD,NVSD,NDSD,NBOTD
INTEGER NRANSWK, K
REAL AOA, PIVOTX, PIVOTY, TUNPARAM, USL, DSL, PHI1, PHI2
REAL CHORDL,FULLTHK,CAMBMX, RADLE
REAL RESLE, RESMID, RESTE, RESWALL,RESBL,YUPWALL,YBOTWALL
REAL TEMP2, TEMP1,TEMP3,TEMP4,PACKFACTOR,REYNOLD
REAL TOL,E1,RF
CHARACTER FIN*20, PLOTFL1*20, MESHFILE*20, RESTFILE*20
CHARACTER FOILGEO*20, FOILDES*40
CHARACTER CDUM*40, QUERY*10, JUNKTEXT*40
CHARACTER PROMPT2*34
REAL PI,TWOPI,ZERO,ONE,HALF
PARAMETER (PI=3.14159,TWOPI=6.28318, NI=200, ZERO=0.0, ONE=1.0)
PARAMETER (HALF=0.5)

```

C
C
C

ARRAYS

```

REAL A(3),B(3),C(NI),D(NI),E(NI), XI(NI),YI(NI),THCK(NI)
REAL PRES1X(NI),PRES1Y(NI),PRES2X(NI),PRES2Y(NI)
REAL SUCT1X(NI),SUCT1Y(NI),SUCT2X(NI),SUCT2Y(NI)
REAL PRES1XS(NI),PRES1YS(NI),PRES2XS(NI),PRES2YS(NI)
REAL SUCT1XS(NI),SUCT1YS(NI),SUCT2XS(NI),SUCT2YS(NI)
REAL PRESSUREX(NI), SUCTIONX(NI)
REAL PRESSUREY(NI), SUCTIONY(NI)
REAL UPLINEX(NI),UPLINEY(NI),DOWNX(NI),DOWNY(NI)
REAL DOWNXS(NI),DOWNYS(NI),UPLINEXS(NI),UPLINEYS(NI)
REAL WALL1X(NI),WALL1Y(NI),WALL2X(NI),WALL2Y(NI)
REAL WALL5X(NI),WALL5Y(NI),WALL6X(NI),WALL6Y(NI)
REAL WALL3X(NI),WALL3Y(NI),WALL4X(NI),WALL4Y(NI)
REAL VERT1X(NI),VERT1Y(NI),VERT3X(NI),VERT3Y(NI)
REAL VERT2X(NI),VERT2Y(NI),VERT4X(NI),VERT4Y(NI)
REAL VERT5X(NI),VERT5Y(NI),VERT6X(NI),VERT6Y(NI)
REAL VERT7X(NI),VERT7Y(NI),VERT8X(NI),VERT8Y(NI)
REAL ZONE1X(NI,NI),ZONE1Y(NI,NI),ZONE2X(NI,NI),ZONE2Y(NI,NI)
REAL ZONE3X(NI,NI),ZONE3Y(NI,NI),ZONE4X(NI,NI),ZONE4Y(NI,NI)
REAL ZONE5X(NI,NI),ZONE5Y(NI,NI),ZONE6X(NI,NI),ZONE6Y(NI,NI)
REAL DZONE5X(NI,NI),DZONE5Y(NI,NI),DZONE6X(NI,NI),DZONE6Y(NI,NI)
REAL DZONE3X(NI,NI),DZONE3Y(NI,NI),DZONE2X(NI,NI),DZONE2Y(NI,NI)
REAL RANSWKPTS(2,NI)
INTEGER IC(6,8,4)

```

C*****

C
C
C
C
C
C
C
C
C
C

Open Control file fname.ctrl: default filemanme is foil2d.ctrl

Seek Parameters describes above, use info for later output file.

OPEN INPUT FILE

```

PROMPT2 = 'PROGRAM CONTROL FILE'/' ('/'/'foil2d.ctrl'/'/'') = '
WRITE (*,'(A,$)') PROMPT2
READ (*,'(A)') FIN
IF (FIN(1:1).EQ.' ') FIN = 'foil2d.ctrl'
WRITE (*,'(A)') 'OPENING FILE: ' // FIN
OPEN (UNIT=4,FILE=FIN,STATUS='OLD')

```

C
C

```

C READ IN INPUT DATA
  READ(4, '(A)') CDUM
  READ(4, '(A)') JUNKTEXT
  READ(4, '(A)') FOILGEO
  READ(4, '(A)') JUNKTEXT
  READ(4, *) AOA, PIVOTX, PIVOTY
  READ(4, '(A)') JUNKTEXT

C
C Tunnel param for setting wall distance
C
  READ(4, *) TUNPARAM
  READ(4, '(A)') JUNKTEXT
  READ(4, *) USL, DSL, PHI1, PHI2
  READ(4, '(A)') JUNKTEXT
  READ(4, *) NUS, NDS, NVS, NTOP, NBOT
  READ(4, '(A)') JUNKTEXT
  READ(4, *) RESLE, RESMID, RESTE, RESWALL, RESBL, PACKFACTOR
  READ(4, '(A)') JUNKTEXT
  READ(4, *) REYNOLD
  READ(4, '(A)') JUNKTEXT
  READ(4, '(A)') MESHFILE
  READ(4, '(A)') JUNKTEXT
  READ(4, '(A)') RESTFILE
  READ(4, '(A)') JUNKTEXT
  READ(4, *) NUMIT
  READ(4, '(A)') JUNKTEXT
  READ(4, *) TOL
  READ(4, '(A)') JUNKTEXT

C
C Adding capability to read RANS wake points
C READ(4, *) NRANSWK
C
  IF (NRANSWK.EQ.ZERO) THEN
    WRITE(*, *) ' *** '
    WRITE(*, *) '***NO RANS WAKE DATA, PROCEED WITH VLM***'
    WRITE(*, *) ' *** '
    ENDIF

C
C
C IF (NRANSWK.LT.ZERO) THEN
  WRITE(*, *) ' *** '
  WRITE(*, *) '*****STRAIGHT WAKE SELECTED*****'
  WRITE(*, *) '*****NO VORTEX LATTICE SOLN AVAIL*****'
  WRITE(*, *) ' *** '
  ENDIF

C
C Test to see if there are points
C IF(NRANSWK .GT.ZERO) THEN
  WRITE(*, *) ' *** '
  WRITE(*, *) '*****READING IN RANS WAKE DATA*****'
  WRITE(*, *) '*****NO VORTEX LATTICE SOLN AVAIL*****'
  WRITE(*, *) ' *** '
  READ(4, *) ((RANSWKPTS(1,k), RANSWKPTS(2,k)), K=1, NRANSWK)
C write(*, *) ((RANSWKPTS(1,k), RANSWKPTS(2,k)), K=1, NRANSWK)
  ENDIF

C
C *****
C *****END OF INPUT FILE READS *****
C *****
C
C OPEN ECHO FILE TO VALIDATE INPUT FILE
C
C PROMPT2 = 'CONTROL CHECK FILE'// ' ( '//' 'check.dat'// ' ) = '
C WRITE (*, '(A,$)') PROMPT2
C READ (*, '(A)') FIN

```

```

C      IF (FIN(1:1).EQ.' ') FIN = 'check.dat'
C      WRITE (*,'(A)') 'OPENING FILE: ' // FIN
C      OPEN (UNIT=3,FILE=FIN,STATUS='UNKNOWN')
C
C      WRITE(3,*)CDUM
C      WRITE(3,*)FOILGEO
C      WRITE(3,*)AOA, PIVOTX, PIVOTY
C      WRITE(3,*)TUNPARAM
C      WRITE(3,*)USL,DSL,PHI1,PHI2
C      WRITE(3,*)NUS, NDS, NVS, NTOP, NBOT,RESLE, RESMID, RESTE, RESWALL
C      WRITE(3,*)RESBL,PACKFACTOR,REYNOLD
C      CLOSE(3)
C      CLOSE(4)
C*****
C
C      OPEN FOIL DATA FILE FOR BASELINE GEOMETRY
C
C
C      WRITE (*,'(A)') 'OPENING GEOMETRY FILE: ' // FOILGEO
C      OPEN (UNIT=7,FILE=FOILGEO,STATUS='OLD')
C      READ IN INPUT DATA
C      READ(7,'(A)')FOILDES
C
C      NFTYPE SPECIFIES FOIL TYPE
C      2 = X,Y GEOMETRY
C      1 = NACA TYPE FORMAT
C
C      READ(7,*)NFTYPE
C      READ(7,*)CHORDL
C      READ(7,*)NPOINTS
C
C      READ IN X,Y DATA
C
C      IF (NFTYPE.EQ.2)THEN
C          DO 5 I=1,NPOINTS
C              READ(7,*)XI(I),YI(I)
C          5      CONTINUE
C      ENDIF
C
C      READ IN NACA FORMATTED DATA
C
C      IF (NFTYPE.EQ.1)THEN
C          READ(7,*)FULLTHK,CAMBMX
C          DO 6 I=1,NPOINTS
C              READ(7,*)XI(I),THCK(I),YI(I)
C          6      CONTINUE
C          READ(7,*)RADLE
C      ENDIF
C      CLOSE(7)
C
C
C*****
C
C      IF THE FOIL IS A NACA TYPE, IT NEEDS TO BE CONVERTED TO
C      X AND Y DATA FOR FURTHER USE IN THE PROGRAM AND FEEDING
C      IN TO INMESH.
C
C      CHECK TO SEE IF IT IS A NACA FOIL, IF SO GO TO CONVERSION
C      SUBROUTINE.
C
C      IF (NFTYPE.EQ.1) THEN
C          CALL NACACONV(NPOINTS,XI,YI,THCK,RADLE)
C      ENDIF

```

```

C
C
C OPEN ECHO FILE TO VALIDATE FOIL GEOMETRY FILE
C
C PROMPT2 = 'GEO VALIDATION FILE'// ' ('// 'foilgeo.dat'//') = '
C WRITE (*,'(A,$)') PROMPT2
C READ (*,'(A)') FIN
C IF (FIN(1:1).EQ.' ') FIN = 'foilgeo.dat'
C WRITE (*,'(A)') 'OPENING FILE: ' // FIN
C OPEN (UNIT=8,FILE=FIN,STATUS='UNKNOWN')
C
C
C WRITE(8,*)FOILDES
C WRITE(8,*)NFTYPE
C WRITE(*,*) 'input debugger', CHORDL,NPOINTS
99 FORMAT(F8.4,F8.4)
C DO 7 I=1,NPOINTS
C WRITE(8,99)XI(I),YI(I)
C 7 CONTINUE
C WRITE(8,*)CDUM
C CLOSE(8)
C*****
C
C NORMALIZE FOIL DIMENSIONS IF REQUIRED
C IF (CHORDL.NE.ONE) THEN
C CALL NORMFOIL(CHORDL,NPOINTS,XI,YI)
C ENDIF
C
C
C*****
C SEND FOIL GEOMETRY TO BE ROTATED FOR ANGLE OF ATTACK
C
C
C IF (AOA.NE.ZERO)THEN
C CALL ROTANGLE(NPOINTS,XI,YI,AOA,PIVOTX,PIVOTY)
C ENDIF
C
C
C IF (AOA.EQ.ZERO)THEN
C WRITE(*,*) 'AOA IS ZERO, ORIGIN DEFAULTS TO LEADING EDGE.'
C ENDIF
C*****
C PREPARE DATA FILE FOR TECPLOT TO VERIFY FOIL ROTATION
C
C CHECK IF USER WANTS CHECK FILE:
C WRITE(*,'(A,$)') 'CREATE A TECPLOT FILE TO VIEW ROTATION?<n>:'
C READ(*,'(A)') QUERY
C
C debugger
C write(*,*) query
C
C IF ((QUERY.EQ.'Y').OR.(QUERY.EQ.'y')) THEN
C PROMPT2 = 'FOIL TECPLOT FILE'// ' ('// 'rotate.plt'//') = '
C WRITE (*,'(A,$)') PROMPT2
C READ (*,'(A)') FIN
C IF (FIN(1:1).EQ.' ') FIN = 'rotate.plt'
C WRITE (*,'(A)') 'OPENING FILE: ' // FIN
C OPEN (UNIT=9,FILE=FIN,STATUS='UNKNOWN')
C WRITE(9,*) 'VARIABLES = X,Y'
C WRITE(9,*) 'ZONE T="Input Geometry"'
C DO 8 I=1,NPOINTS
C WRITE(9,*)XI(I),YI(I)

```

```

8      CONTINUE
      WRITE(9,*) 'ZONE T="Reference Line"'
      WRITE(9,*)-1.0,zero
      WRITE(9,*)zero,zero
      WRITE(9,*)ONE,ZERO
c      CLOSE(9)
      ENDIF
C*****
C      SPLIT FOIL IN TO FOUR ARCS.  SPLINE THE ARCS AND RETURN GRADED
C      POINT ARRAYS TO DEFINE ARCS.  THIS WILL DETERMINE THE MESH DENSITIES
C      FOR EACH ZONAL AREA.
C
C
C      IDENTIFY SPLITTING BOUNDARIES, FOILZONE returns the integer array
c      position of the splitting points.
C
      CALL FOILZONE(XI,NPOINTS,MIDPRES,MIDSUCT,NOSE)
C
C      NOW TAKE THE FOIL OFFSETS AND SPLIT IN TO EIGHT ARRAYS FOR
C      FEEDING IN TO THE FNSPLT SUBROUTINE. 4 each of X & Y points
C
      CALL ARCMaker(XI,YI,1,MIDPRES,PRES1X,PRES1Y)
      CALL ARCMaker(XI,YI,MIDPRES,NOSE,PRES2X,PRES2Y)
      CALL ARCMaker(XI,YI,NOSE,MIDSUCT,SUCT1X,SUCT1Y)
      CALL ARCMaker(XI,YI,MIDSUCT,NPOINTS,SUCT2X,SUCT2Y)
C
C*****
c
C      SEND TO SPLINING ROUTINE
c      Spline First Interval TE to midchord pressure side
C
C      NTEMP is number of points want to get back from fnsplt
      NPRES1=INT((float(NBOT)-1.0)/2.0)
C
C
      CALL FNSPLT(MIDPRES,NPRES1,RESTE,RESMID,PRES1X,PRES1Y,PRES1XS,
+pres1YS)
C
C      debugger
c
c      write(*,*) 'came back from arcmaker ok, nPRES1 is:',NPRES1
c      write(*,*) 'Doing Aft Pressure Side'
C
c      write(79,*) 'ZONE'
c      do 4990 i=1,npres1
c          write(79,*) pres1xs(i),pres1ys(i)
c 4990 continue
C
c      end debugger
c      Now spline from midchord pressure side to LE
c
      NPRES2 = NPRES1+2
c      NTEMP2 is number of points going in to fnsplt
      NTEMP2 = NOSE - MIDPRES +1
      CALL FNSPLT(NTEMP2,NPRES2,RESMID,RESLE,PRES2X,PRES2Y,PRES2XS,
+PRES2YS)
C
C      debugger
c      write(*,*) 'Doing Forw Pressure Side'
c      write(*,*) 'number of points GOING IN TO fnsplt:',ntemp2
c      write(*,*) 'number of points from fnsplt:',npres2
c      write(79,*) 'ZONE'
c      do 5000 i=1,npres2
c          write(79,*) pres2xs(i),pres2ys(i)

```



```

c 5000 continue
c end debugger
c
C *****now do other half of foil.
c   Now spline from LE to midchord suction side
C   NTEMP is number of points want to get back from fnsplt
C
      NSUCT1=INT((float(NTOP)-1.0)/2.0)
C
c
      NTEMP2 = MIDSUCT - NOSE+1
      CALL FNSPLT(NTEMP2,NSUCT1,RESLE,RESMID,SUCT1X,SUCT1Y,SUCT1XS,
+      SUCT1YS)
C      debugger
c
c      write(*,*) 'Doing Forw Suction Side'
c      write(*,*) 'number of points GOING IN TO fnsplt:',ntemp2
c      write(*,*) 'number of points from fnsplt:',nsuct1
c      write(79,*) 'ZONE'
c      do 5010 i=1,nsuct1
c          write(79,*) SUCT1xs(i),SUCT1ys(i)
c 5010 continue
c
c
c      Now spline from Midchord Suction Side to Trailing Edge
c
c
      NSUCT2 = NSUCT1+2
c      NTEMP2 is number of points going in to fnsplt
      NTEMP2 = NPOINTS - MIDSUCT+1
      CALL FNSPLT(NTEMP2,NSUCT2,RESMID,RESTE,SUCT2X,SUCT2Y,SUCT2XS,
+      SUCT2YS)
C      debugger
c      write(*,*) 'Doing AFT Pressure Side'
c      write(*,*) 'number of points GOING IN TO fnsplt:',ntemp2
c      write(*,*) 'number of points from fnsplt:',nsuct2
c      write(79,*) 'ZONE'
c      do 5020 i=1,nsuct2
c          write(79,*) SUCT2xs(i),SUCT2ys(i)
c 5020 continue
c end debugger
c
C*****
C      Rejoin the pairs of arcs and make this the final foil geometry
c
      CALL ARCJOINER(NPRES1,NPRES2,NPRESSURE,PRES1XS,PRES2XS,PRES1YS
+      ,PRES2YS,PRESSUREX,PRESSUREY)
C
      CALL ARCJOINER(NSUCT1,NSUCT2,NSUCTION,SUCT1XS,SUCT2XS,SUCT1YS
+      ,SUCT2YS,SUCTIONX,SUCTIONY)
c
C*****
c      Append the plot file with the splined data
c
c
      IF ((QUERY.EQ.'Y').OR.(QUERY.EQ.'y')) THEN
          WRITE(9,*) 'ZONE T="Pressure Side"'
          DO 9 I=1,NPRESSURE
              WRITE(9,*) PRESSUREX(I),PRESSUREY(I)
          9      CONTINUE
          write(9,*) 'ZONE T="Suction Side"'
          DO 10 I=1,NSUCTION
              WRITE(9,*) SUCTIONX(I),SUCTIONY(I)
          10     CONTINUE

```

```

        CLOSE(9)
      ENDIF
c*****
c    Define upstream geometry of tunnel:
c
c        First do horizontal line extending from leading
c        edge to forward end of tunnel.
      NTEMP3=10
c      write(*,*) ntemp3
      CALL TUNB(USL,SUCTIONX(1),SUCTIONY(1),UPLINEX,UPLINEY,NTEMP3)
      CALL FNSPLT(NTEMP3,NUS,RESLE,RESMID,UPLINEX,UPLINEY,UPLINEXS,
+UPLINEYS)
c*****

c    Define downstream geometry of tunnel:
c        First do horizontal line extending downstream from
c        trailing edge to end of tunnel
c        .....
c
c    Adapt the GRID boundaries to the wake based on VORTEX LATTICE
c    SOLUTION.
c
c      IF(NRANSWK.EQ.ZERO) THEN
c      CALL ADAPT(PRESSUREX,PRESSUREY,SUCTIONX,SUCTIONY,DOWNX,DOWNY,
+      NBOT,NTOP,DSL,NTEMP3,TUNPARAM)
c      ENDIF
c
cc
c
c
c    if ranswk greater than 0 then there is rans data
c
c    IF THE DATA IS BAD THEN NRANSWK WILL BE SET TO -1 AND
c    A STRAIGHT WAKE WILL BE USED.
c
c      IF(NRANSWK.GT.ZERO) THEN
c      CALL RANSWAKE(DSL,PRESSUREX(1),PRESSUREY(1),RANSWKPTS,
+      DOWNX,DOWNY,NTEMP3,NRANSWK)
c      ENDIF
c
c
c    If NRANSWK is set to -1 then it will do a straight wake.
c
c
c      IF(NRANSWK.LT.ZERO) THEN
c      write(*,*)'***'
c      write(*,*)'*****DEFAULTING TO STRAIGHT WAKE*****'
c      WRITE(*,*)'***'
c      CALL TUNB(DSL,PRESSUREX(1),PRESSUREY(1),DOWNX,DOWNY,NTEMP3)
c      ENDIF
c
c      (NIN,NOUT,DS1,DS2,XI,YI,XO,YO)
c
c    Spline downstream line.
c    CALL FNSPLT(NTEMP3,NDS,RESTE,RESMID,DOWNX,DOWNY,DOWNXS,DOWNYS)
c
c
c
c    debugger
c
c      write(79,*) 'ZONE'
c      do 5030 i=1,nus
c      write(79,*) uplinexs(i),uplineys(i)
c 5030 continue
c

```



```

c      TEMP1=UPLINEXS(NUS)
      TEMP2=UPLINEYS(NUS)
      TEMP3=WALL1X(NUS)
      TEMP4=WALL1Y(NUS)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT1X,VERT1Y,NVS,PACKFACTOR)
c
c      Leading Edge vertical (above)
c
c      TEMP1=UPLINEXS(1)
      TEMP2=UPLINEYS(1)
      TEMP3=WALL1X(1)
      TEMP4=WALL1Y(1)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT3X,VERT3Y,NVS,PACKFACTOR)
c
c      Trailing Edge vertical(above)
c
c      TEMP1=downXS(1)
      TEMP2=downYS(1)
      TEMP3=WALL5X(1)
      TEMP4=WALL5Y(1)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT5X,VERT5Y,NVS,PACKFACTOR)
c
c      Downstream Vertical Line (above)
c
c      TEMP1=downXS(nds)
      TEMP2=downYS(nds)
      TEMP3=WALL5X(nds)
      TEMP4=WALL5Y(nds)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT7X,VERT7Y,NVS,PACKFACTOR)
c
c*****DO ALL THE BELOW LINES*****
c
c      Upstream Vertical Line (below)
c
c      TEMP1=UPLINEXS(NUS)
      TEMP2=UPLINEYS(NUS)
      TEMP3=WALL2X(NUS)
      TEMP4=WALL2Y(NUS)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT2X,VERT2Y,NVS,PACKFACTOR)
c
c
c      Leading Edge vertical (below)
c
c      TEMP1=UPLINEXS(1)
      TEMP2=UPLINEYS(1)
      TEMP3=WALL2X(1)
      TEMP4=WALL2Y(1)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT4X,VERT4Y,NVS,PACKFACTOR)
c
c
c      Trailing Edge vertical(below)
c
c      TEMP1=downXS(1)
      TEMP2=downYS(1)
      TEMP3=WALL6X(1)
      TEMP4=WALL6Y(1)
      CALL TUND(TEMP1,TEMP2,TEMP3,TEMP4,RESBL,RESWALL,
+VERT6X,VERT6Y,NVS,PACKFACTOR)

```



```

c 5050 continue
c
c      end debugger

      CALL BORDERS(ZONE2X,ZONE2Y,VERT3X,VERT3Y,SUCTIONX,SUCTIONY
+ ,VERT5X,VERT5Y,WALL3X,WALL3Y,NTOP,NVS,NI)
C
C
C      ZONE THREE
C
C
C      CALL BORDERS(ZONE3X,ZONE3Y,VERT5X,VERT5Y,DOWNXS,DOWNYS,
+ VERT7X,VERT7Y,WALL5X,WALL5Y,NDS,NVS,NI)
C
C
C      ZONE FOUR
C
C      CALL BORDERS(ZONE4X,ZONE4Y,VERT2X,VERT2Y,WALL2X,WALL2Y,
+ VERT4X,VERT4Y,UPLINEXS,UPLINEYS,NUS,NVS,NI)
C
C
C      ZONE FIVE
C
C      CALL BORDERS(ZONE5X,ZONE5Y,VERT4X,VERT4Y,wall4X,wall4Y
+ ,VERT6X,VERT6Y,pressureX,pressureY,NBOT,NVS,NI)
C
C
C      ZONE SIX
C
C      CALL BORDERS(ZONE6X,ZONE6Y,VERT6X,VERT6Y,WALL6X,WALL6Y,
+ VERT8X,VERT8Y,DOWNXS,DOWNYS,NDS,NVS,NI)
C
C*****
C      Open Tecplot File for viewing Mesh.
c
      PROMPT2 = 'Mesh View TECPLOT FILE'// ' ('// 'mesh.plt'//') = '
      WRITE (*,'(A,$)') PROMPT2,':'
      READ (*,'(A)') FIN
      IF (FIN(1:1).EQ.' ') FIN = 'mesh.plt'
      WRITE (*,'(A)') 'OPENING FILE: ' // FIN
      OPEN (UNIT=82,FILE=FIN,STATUS='UNKNOWN')

c      topzones
      write(82,*) 'VARIABLES =X,Y'
      write(82,*) 'ZONE T="ZONE 1" I=','Nus',' J=','NVS',' F=POINT'
      do 6000 j=1,nvs
        do 6010 i=1,nus
          write(82,*) zone1x(i,j),zone1y(i,j)
6010      continue
6000      continue
      write(82,*) 'ZONE T="ZONE 2" I=','Ntop',' J=','NVS',' F=POINT'
      do 6020 j=1,nvs
        do 6030 i=1,ntop
          write(82,*) zone2x(i,j),zone2y(i,j)
6030      continue
6020      continue
      write(82,*) 'ZONE T="ZONE 3" I=','Nds',' J=','NVS',' F=POINT'
      do 6040 j=1,nvs
        do 6050 i=1,nds
          write(82,*) zone3x(i,j),zone3y(i,j)
6050      continue
6040      continue
c

```

```

c      bottom xones
c
      write(82,*)'ZONE T="ZONE 4" I=',Nus,' J=',NVS,' F=POINT'
      do 6060 j=1,nvs
        do 6070 i=1,nus
          write(82,*) zone4x(i,j),zone4y(i,j)
6070      continue
6060      continue
      write(82,*)'ZONE T="ZONE 5" I=',Nbot,' J=',NVS,' F=POINT'
      do 6080 j=1,nvs
        do 6090 i=1,nbot
          write(82,*) zone5x(i,j),zone5y(i,j)
6090      continue
6080      continue
      write(82,*)'ZONE T="ZONE 6" I=',Nds,' J=',NVS,' F=POINT'
      do 7000 j=1,nvs
        do 7010 i=1,nds
          write(82,*) zone6x(i,j),zone6y(i,j)
7010      continue
7000      continue
      close(82)
c
c*****
c      Generate INMESH input file:
c
cC      CHECK IF USER WANTS CHECK FILE:
c
c
      write(*,*) '#####'
      write(*,*) ' '
      write(*,*) 'View the grid on TECPLOT, Check boundaries for'
      write(*,*) 'wrapping. If the grid is not correct or if'
      write(*,*) 'you want to make changes, you may abort gen-'
      write(*,*) 'eration of the INMESH file.'
      write(*,*) ' '
      write(*,*) '#####'
      write(*,*) ' '

      WRITE(*,'(A,$)') 'DO YOU WANT TO WRITE THE INMESH FILE?<n>:'
      READ(*,'(A)') QUERY

C
C      debugger
c      write(*,*) query
c
      IF ((QUERY.EQ.'Y').OR.(QUERY.EQ.'y')) THEN
C
C
      WRITE (*,'(A)') 'WRITING FILE: ' // MESHFILE
      OPEN (UNIT=80,FILE=MESHFILE,STATUS='UNKNOWN')

C
C      FORMAT FIRST LINE
C
C      E1 is the orthogonality at boundry
      E1=1.0
C
C      Relaxation parameter
      RF=0.5
C
C      Tell INMESH it is a restart
      NRESTART=2
c
c      Unknown parameter
      NWHAT=1
C

```

```

C    WRITE FIRST LINE OF THE INMESH FILE
98  format(f4.2,f12.8,i7,f8.4,2i4)
97  format(i3,7i3)
96  format(i3,2i5,a20)
    if(numit.eq.1)then
      nrestart=1
    endif
    write(80,98) E1,TOL,NUMIT,RF,NRESTART,NWHAT

c
c
c*****DO THE FIRST THREE ZONES*****
C
c*****Write out Boundaries for Zone 1
c
c    write(80,96)1,nus,nvs,'    ***Zone 1***'
c
c    Zone 1 Line 1
c    NLINE=1
c    DATA (IC(1,I,1),I=1,8)/1,5,0,0,1,1,0,0/
c    write(80,97) (IC(1,I,NLINE),I=1,8)
c
c    call zonelines(zone1x,zone1y,nus,nvs,NLINE,80)
C
c    Zone 1 Line 2
c    NLINE=2
c    DATA (IC(1,I,2),I=1,8)/2,1,0,0,1,2,0,0/
c    write(80,97) (IC(1,I,NLINE),I=1,8)
c
c    call zonelines(zone1x,zone1y,nus,nvs,NLINE,80)
c
c    Zone 1 Line 3
c    NLINE=3
c    DATA (IC(1,I,3),I=1,8)/3,0,2,1,1,1,0,0/
c    write(80,97) (IC(1,I,NLINE),I=1,8)
c
c    Zone 1 Line 4
c    NLINE=4
c    DATA (IC(1,I,4),I=1,8)/4,6,0,0,1,4,0,0/
c    write(80,97) (IC(1,I,NLINE),I=1,8)
c
c    call zonelines(zone1x,zone1y,nus,nvs,NLINE,80)
Cc
c
c*****Zone 2
c    Write out Boundaries for Zone 2
c
c    write(80,96)2,ntop+1,nvs,'    ***Zone 2***'
c
c    Zone 2 Line 1
c    NLINE=1
c    DATA (IC(2,I,1),I=1,8)/1,0,1,3,1,1,0,0/
c    write(80,97) (IC(2,I,NLINE),I=1,8)
c
c
c    Zone 2 Line 2
c    NLINE=2
c    DATA (IC(2,I,2),I=1,8)/2,6,0,0,2,2,0,0/
c    write(80,97) (IC(2,I,NLINE),I=1,8)
c
c    write(80,*)zone1x(nus-1,1),zone1y(nus-1,1)
c    call zonelines(zone2x,zone2y,ntop,nvs,NLINE,80)
c
c    Zone 2 Line 3
c    NLINE=3
c    DATA (IC(2,I,3),I=1,8)/3,0,3,1,1,1,0,0/

```



```

        write(80,97) (IC(2,I,NLINE),I=1,8)
c
c   Zone 2 Line 4
c   NLINE=4
c   DATA (IC(2,I,4),I=1,8)/4,6,0,0,2,4,0,0/
c   write(80,97) (IC(2,I,NLINE),I=1,8)
c
c   write(80,*)zone1x(nus-1,nvs),zone1y(nus-1,nvs)
c   call zonelines(zone2x,zone2y,ntop,nvs,NLINE,80)
Cc
c
c
c   c*****Zone 3
c   Write out Boundaries for Zone 3
c
c   write(80,96)3,nds+1,nvs,'   ***Zone 3***'
c
c   Zone 3 Line 1
c   NLINE=1
c   DATA (IC(3,I,1),I=1,8)/1,0,2,3,1,1,0,0/
c   write(80,97) (IC(3,I,NLINE),I=1,8)
c
c
c   Zone 3 Line 2
c   NLINE=2
c   DATA (IC(3,I,2),I=1,8)/2,1,0,0,3,2,0,0/
c   write(80,97) (IC(3,I,NLINE),I=1,8)
c
c   write(80,*)zone2x(ntop-1,1),zone2y(ntop-1,1)
c   call zonelines(zone3x,zone3y,nds,nvs,NLINE,80)
c
c   Zone 3 Line 3
c   NLINE=3
c   DATA (IC(3,I,3),I=1,8)/3,2,0,0,3,3,0,0/
c   write(80,97) (IC(3,I,NLINE),I=1,8)
c
c   call zonelines(zone3x,zone3y,nds,nvs,NLINE,80)
c
c   Zone 3 Line 4
c   NLINE=4
c   DATA (IC(3,I,4),I=1,8)/4,5,0,0,3,4,0,0/
c   write(80,97) (IC(3,I,NLINE),I=1,8)
c
c   write(80,*)zone2x(ntop-1,nvs),zone2y(nus-1,nvs)
c   call zonelines(zone3x,zone3y,nds,nvs,NLINE,80)
Cc%%%%%%%%%%%%%%
c
c*****DO THE SECOND THREE ZONES*****
c
c*****Write out Boundaries for Zone 4
c
c   write(80,96)4,nus,nvs,'   ***Zone 4***'
c
c   Zone 4 Line 1
c   NLINE=1
c   DATA (IC(4,I,1),I=1,8)/1,5,0,0,4,1,0,0/
c   write(80,97) (IC(4,I,NLINE),I=1,8)
c
c   call zonelines(zone4x,zone4y,nus,nvs,NLINE,80)
c
c   Zone 4 Line 2
c   NLINE=2
c   DATA (IC(4,I,2),I=1,8)/2,6,0,0,4,2,0,0/
c   write(80,97) (IC(4,I,NLINE),I=1,8)

```

```

c      call zonelines(zone4x,zone4y,nus,nvs,NLINE,80)
c
c      Zone 4 Line 3
c      NLINE=3
c      DATA (IC(4,I,3),I=1,8)/3,0,5,1,4,1,0,0/
c      write(80,97) (IC(4,I,NLINE),I=1,8)
c
c      Zone 4 Line 4
c      NLINE=4
c      DATA (IC(4,I,4),I=1,8)/4,1,0,0,4,4,0,0/
c      write(80,97) (IC(4,I,NLINE),I=1,8)
c
c      call zonelines(zone4x,zone4y,nus,nvs,NLINE,80)
Cc
c
c*****Zone 5
c      Write out Boundaries for Zone 5
c
c      write(80,96)5,nbot+1,nvs,'    ***Zone 5***'
c
c      Zone 5 Line 1
c      NLINE=1
c      DATA (IC(5,I,1),I=1,8)/1,0,4,3,4,1,0,0/
c      write(80,97) (IC(5,I,NLINE),I=1,8)
c
c
c      Zone 5 Line 2
c      NLINE=2
c      DATA (IC(5,I,2),I=1,8)/2,6,0,0,5,2,0,0/
c
c      write(80,97) (IC(5,I,NLINE),I=1,8)
c
c      write(80,*) zone4x(nus-1,1),zone4y(nus-1,1)
c      call zonelines(zone5x,zone5y,ntop,nvs,NLINE,80)
c
c      Zone 5 Line 3
c      NLINE=3
c      DATA (IC(5,I,3),I=1,8)/3,0,6,1,4,1,0,0/
c      write(80,97) (IC(5,I,NLINE),I=1,8)
c
c      Zone 5 Line 4
c      NLINE=4
c      DATA (IC(5,I,4),I=1,8)/4,6,0,0,5,4,0,0/
c      write(80,97) (IC(5,I,NLINE),I=1,8)
c
c      write(80,*) zone4x(nus-1,nvs),zone4y(nus-1,nvs)
c      call zonelines(zone5x,zone5y,ntop,nvs,NLINE,80)
Cc
c
c
c*****Zone 6
c      Write out Boundaries for Zone 6
c
c      write(80,96)6,nds+1,nvs,'    ***Zone 6***'
c
c      Zone 6 Line 1
c      NLINE=1
c      DATA (IC(6,I,1),I=1,8)/1,0,5,3,4,1,0,0/
c      write(80,97) (IC(6,I,NLINE),I=1,8)
c
c
c      Zone 6 Line 2
c      NLINE=2
c      DATA (IC(6,I,2),I=1,8)/2,6,0,0,6,2,0,0/

```

```

      write(80,97) (IC(6,I,NLINE),I=1,8)
C
      write(80,*)zone5x(nbot-1,1),zone5y(nbot-1,1)
      call zonelines(zone6x,zone6y,nds,nvs,NLINE,80)
C
C   Zone 6 Line 3
      NLINE=3
      DATA (IC(6,I,3),I=1,8)/3,2,0,0,6,3,0,0/
      write(80,97) (IC(6,I,NLINE),I=1,8)
C
      call zonelines(zone6x,zone6y,nds,nvs,NLINE,80)
C   Zone 6 Line 4
      NLINE=4
      DATA (IC(6,I,4),I=1,8)/4,1,0,0,6,4,0,0/
      write(80,97) (IC(6,I,NLINE),I=1,8)
C
      write(80,*) zone5x(nbot-1,nvs),zone5y(nbot-1,nvs)
      call zonelines(zone6x,zone6y,nds,nvs,NLINE,80)
C
      CLOSE(80)
C*****
C   IN ORDER TO WRITE THE RESTART FILE, ZONES 2,3,5,6 NEED TO
C   BE APPENDED WITH THE "N-1" COLUMN FROM THE UPSTREAM ARRAY
C   SO THAT OVERLAPPING CONTINUITY CAN BE MAINTAINED
C   Append zone 2
      CALL MAKEDUM(ZONE1X,ZONE1Y,NUS,NVS,ZONE2X,ZONE2Y,NTOP,NVS,
+DZONE2X,DZONE2Y,NTOPD,NVSD)
C   Append zone 3
      CALL MAKEDUM(ZONE2X,ZONE2Y,NTOP,NVS,ZONE3X,ZONE3Y,NDS,NVS,
+DZONE3X,DZONE3Y,NDSD,NVSD)
C   Append zone 5
      CALL MAKEDUM(ZONE4X,ZONE4Y,NUS,NVS,ZONE5X,ZONE5Y,NBOT,NVS,
+DZONE5X,DZONE5Y,NBOTD,NVSD)
C   Append zone 6
      CALL MAKEDUM(ZONE5X,ZONE5Y,NBOT,NVS,ZONE6X,ZONE6Y,NDS,NVS,
+DZONE6X,DZONE6Y,NDSD,NVSD)
C
C*****
C   WRITE INMESH RESTART FILE
C   IF (NUMIT.GT.1)THEN
      WRITE (*,'(A)') 'WRITING FILE: ' // RESTFILE
      OPEN (UNIT=78,FILE=RESTFILE,FORM='UNFORMATTED',
+      STATUS='UNKNOWN')
      NZONE=6
      WRITE(78)NZONE
C   ****ZONE 1*****
      NZONE=1
      WRITE(78)NUS,NVS
      WRITE(78) ((ZONE1X(I,J),ZONE1Y(I,J),I=1,NUS),J=1,NVS)
      WRITE(78) ((IC(NZONE,M,N),M=1,8),N=1,4)
C
C   ****ZONE 2*****
      NZONE=2
      WRITE(78)NTOPD,NVSD
      WRITE(78) ((DZONE2X(I,J),DZONE2Y(I,J),I=1,NTOPD),J=1,NVSD)
      WRITE(78) ((IC(NZONE,M,N),M=1,8),N=1,4)
C
C   ****ZONE 3*****
      NZONE=3
      WRITE(78)NDSD,NVSD
      WRITE(78) ((DZONE3X(I,J),DZONE3Y(I,J),I=1,NDSD),J=1,NVSD)
      WRITE(78) ((IC(NZONE,M,N),M=1,8),N=1,4)
C

```



```

        write(*,*)'*****'
        write(*,*)'ERROR: RANS wake extends past flow domain'
        write(*,*)'*****'
        NUMIN=-1
    ENDIF

C
C   IF THE DATA IS OK THEN PROCEED THROUGH THIS:
C   IF (NUMIN.GT.0)THEN
C
C       FIRST THE LINE NEED TO BE FIXE UP A LITTLE.
C       THE BEGINNING NEEDS TO START AT THE TRAILING EDGE
C       OF THE FOIL AND THE END NEED TO BE AT THE DOWN
C       STREAM LIMIT OF THE FOIL
C
C       DO 100, I=2,NUMIN+1
C           XTEMP(I)=XYIN(1,I-1)
C           YTEMP(I)=XYIN(2,I-1)
100    CONTINUE
C
C       PUT BEGINNING AT THE TRAILING EDGE
C       XTEMP(1)=XTE
C       YTEMP(1)=YTE
C
C       PUT THE END AT THE DOWNSTREAM LIMIT
C       XTEMP(NUMIN+2)=TEMP1
C       YTEMP(NUMIN+2)=YTEMP(NUMIN+1)
C
C       SPLINE THE LINE AND GET THE CUBIC COEFFICIENTS
C       CALL UGLYDK(NUMIN+2,1,1,XTEMP,YTEMP,0,0,WAKCUB)
C
C       DEBUGGER
C       WRITE(*,*) 'ONE OF THE CUBICS IS:',WAKCUB(5)
C       WE WANT TO SEND NOUT X,Y POINTS BACK TO MAIN
C       DSL IS HOW FAR TO MARCH TOTAL
C       TEMP2=DSL/(FLOAT(NOUT)-1.0)
C       DEBUGGER
C       WRITE(*,*) 'INTERVAL IS:',TEMP2
C       XOUT(1)=XTE
C
C       DO 200 I=2,NOUT
C           XOUT(I)=XOUT(I-1)+TEMP2
C       DEBUGGER
C       WRITE(*,*)'THE XOUT VALUE IS:',XOUT(I)
200    CONTINUE
C
C       NOW GET THE CORRESPONDING Y VALUES
C       CALL EVALDK(NUMIN+2,NOUT,XTEMP,XOUT,YOUT,WAKCUB)
C
C
C       WRITE(*,*)'***'
C       WRITE(*,*)'***RANS DATA INCORPORATED IN TO GRID***'
C       WRITE(*,*)'***'
C       WRITE(*,*)NUMIN,NOUT
C       WRITE(*,*) ((XOUT(k),YOUT(k)),K=1,NOUT)
C   ENDIF
C
C   RETURN
C   END
C
C+++++
C
C   This subroutine uses a two-d vortex lattice lifting line to

```

```

c      determine find the wake dividing line behind an arbitrary
c      foil section. This subroutine is a generalization of the
c      VLM2D code presented in the 13.04 course notes.
C
      SUBROUTINE ADAPT(PRESSX,PRESSY,SUCTX,SUCTY,WAKEX,WKEY,
+      NPRES,NSUCT,DSL,NWAKE,TUNPARAM)
      IMPLICIT NONE
      INTEGER NPRES,NSUCT,NWAKE,I,J,K,N,M,N,NPTS,NPAN,IERR,NTOT
      INTEGER NIMAGEPR
      REAL PI
      PARAMETER (NI=200,PI=3.141592653589793E00)
      REAL DSL,DELS,LEN,DX,DY,TOP,sum,ENDX,U,V,TEMP,STEP
      REAL TUNPARAM, RTEMP, DXR, DYR, YIMAGE,signimage

C
C      ARRAYS
C
      REAL PRESSX(NI),PRESSY(NI),SUCTX(NI),SUCTY(NI)
      REAL WAKEX(NI),WKEY(NI),cambx(ni),camby(ni)
      REAL TOPCUBX(4*NI),BOTCUBX(4*NI),CAMBCUBX(4*NI)
      REAL TOPCUBY(4*NI),BOTCUBY(4*NI),CAMBCUBY(4*NI)
      REAL WAKCUB(4*NI)
      REAL ARCTOP(NI),ARCBOT(NI),STEMP(NI),ARCAMB(NI)
      REAL TEMPTX(NI),TEMPTY(NI)
      REAL TEMPBX(NI),TEMPBY(NI)
      REAL SV(NI),SC(NI),DS(NI)
      REAL XV(NI),YV(NI),XC(NI),YC(NI),B(NI),TX(NI),TY(NI)
      REAL XN(NI),YN(NI),A(NI,NI),GAMMA(NI),WKAREA(NI),dydx(ni)
      INTEGER IPIVOT(NI)

C
C      What is the Y distance to the first image set? )
c      (the "Image Plane" is at half this distance
c      YIMAGE=1/(TUNPARAM)

C
c      write(*,*)'the wall is at:' , ywall
C      Spline Foil Offsets and determine cubic coefficients.
C
C      Array ARC___ is returned with arclength params non-dimed from 0..1
C
      CALL PUGLYDK(NSUCT,SUCTX,SUCTY,ARCTOP,TOPCUBX,TOPCUBY)
      CALL PUGLYDK(NPRES,PRESSX,PRESSY,ARCBOT,BOTCUBX,BOTCUBY)

C
C      Extract Points along the top and bottom surface to find the
c      mean camber line.
C
C      HOW MANY POINTS TO EXTRACT?
      NPTS=40

C
C
C      AT WHAT ARC COORDINATE LOCATIONS TO EXTRACT?
C
c      write(*,*) arctop(nsuct)

      STEMP(1)=0.0
      DO 100 I=1,NPTS
        STEMP(I)=(FLOAT(I-1)/FLOAT(NPTS-1))*ARCTOP(NSUCT)
c        write(*,*)stemp(i)
100  CONTINUE

C
C      EXTRACT THE VALUES ON THE SUCTION SIDE:
C
      CALL PEVALDK(NSUCT,NPTS,STEMP,ARCTOP,TEMPTX,TEMPTY,TOPCUBX
+      ,TOPCUBY)
c

```

```

C
C   AT WHAT ARC COORDINATE LOCATIONS?
C   STEMP(1)=0.0

      DO 200 I=1,NPTS
        STEMP(I)=(FLOAT(I-1)/FLOAT(NPTS-1))*ARCBOT(NPRES)
200  CONTINUE
C
C   EXTRACT THE VALUES ON THE PRESSURE SIDE:
C
C   CALL PEVALDK(NPRES,NPTS,STEMP,ARCBOT,TEMPBX,TEMPBY,BOTCUBX
+,BOTCUBY)
C
C   FIND THE MEAN CAMBER LINE(SIMPLE MEAN OF TWO COORDINATES):
C
      DO 300 I=1,NPTS
        CAMBX(I)=(TEMPTX(I)+TEMPBX(NPTS-I+1))*0.50
        CAMBY(I)=(EMPTY(I)+TEMPBY(NPTS-I+1))*0.50
C
C   debugger
C   WRITE(88,*)CAMBX(I),CAMBY(I)
300  CONTINUE
C   debugger
C
C   Spline the mean camber line to obtain the Cubic Coefficients
C   Once again using the arc parameter scaling.
C   CALL PUGLYDK(NPTS,CAMBX,CAMBY,ARCAMB,CAMBCUBX,CAMBCUBY)
C
C   DETERMINE THE LOCATIONS OF THE VORTICES AND CONTROL POINTS
C
C   How many vortices and control points should there be?
C   A convergence study was performed by varying the number
C   of panels. The difference in lift coefficient is less
C   than 0.1% when stepping from 20 to 40 panels. So. . .
C   20 panels is sufficient.
C   NPAN=40
C
C   How many pairs of image foils should there be?
C   Obviously, the images should only be put in in pairs.
C   This value will be adjusted until there is a converged
C   lift coefficient.
C
C   A convergence study was conducted. The relative error is
C   approx 0.04% with 16 image pairs. This is sufficient
C   for this application.
C   NIMAGEPR=16
C
C   Establish COSINE spacing on ARC Length Parameter
C   this is straight out of the 13.04 notes.
C   DELS is the COSINE spacing interval
C   DELS=PI/NPAN

      DO 400 I=1,NPAN
        SV(I)=0.50*(1.0-COS((I-0.50)*DELS))
        SC(I)=0.50*(1.0-COS(I*DELS))
        DS(I)=PI*SQRT(SV(I)*(1.0-SV(I)))/NPAN
400  CONTINUE
C
C   EXTRACT THE X AND Y VALUES OF THE VORTICES AND CONTROL POINTS
C
C   CALL PEVALDK(NPTS,NPAN,SV,ARCAMB,XV,YV,CAMBCUBX
+,CAMBCUBY)
C   CALL PEVALDK(NPTS,NPAN,SC,ARCAMB,XC,YC,CAMBCUBX
+,CAMBCUBY)
C

```

```

c      To Panel the walls insert that stuff here:
c
c      How far away are the walls:  TUNPARAM dictates

c      How far do you want to panel up an down stream of
c      the foil?  DSL is already passed in.  So a reasonable
c      limit for paneling is LE - DSL to TE + DSL + 1
c
c      What is a reasonable spacing:  UNIFORM

c      How many panels per WALL:  2*NPAN seems OK

c
c      write locations out to datafile for inspection to fort.88
c      debugger
c      WRITE(88,*)'VARIABLES=XV,YV,XC,YC'
c      WRITE(88,*) 'ZONE T=VORT'
c      DO 500 I=1,NPAN
c          WRITE(88,*)XV(I),YV(I),XC(I),YC(I)
500  CONTINUE

C
c      Calculate the UNIT NORMAL XN,YN at each control point
c      unit normal not used for anything yet, but it may be
c      useful in the future.
c
c      Also, slope is required on the foil for V*n equation later
c
c      write(88,*)'zone'
c      DO 600 I=1,NPAN-1
c          DX=XV(I+1)-XV(I)
c          DY=YV(I+1)-YV(I)
c          LEN=SQRT(DX*DX+DY*DY)
c          XN(I)=-DY/LEN
c          YN(I)=DX/LEN
c          dydx(i)=dy/dx
c          write(88,*) xn(i),yn(i)
600  CONTINUE

C
C      FIX UP THE VALUE AT THE TRAILING EDGE BECAUSE THERE IS NO
C      VORTEX AFTER THE LAST CONTROL POINT.
C
c      DX=XC(NPAN)-XV(NPAN)
c      DY=YC(NPAN)-YV(NPAN)
c      XN(NPAN)=-DY/SQRT(DX*DX+DY*DY)
c      YN(NPAN)=DX/SQRT(DX*DX+DY*DY)
c      dydx(npan)=dy/dx
c      write(88,*)xn(npan),yn(npan)
C
c      ALL THE UNIT NORMALS & Slopes ARE NOW ESTABLISHED.
C
C      COMPUTE THE INFLUENCE COEFFICIENTS A(N,M) AND THEN
C      INVERT THE MATRIX.(linearized)
C      The A matrix are the influence coefficients.  Based on
c      poor agreement with RANS, it was decided to go to the
c      true locations of the vortices on the mean camber line
c      vice a linearized surface.
c
c
c
c      TOP=1.0/(2.0*PI)
c      DO 700 N=1,NPAN
c      DO 700 M=1,NPAN
c          rtemp=sqrt((xv(m)-xc(n))**2+(yv(m)-yc(n))**2)
c          dyr=-(xc(n)-xv(m))/rtemp
c          dxr=-(yc(n)-yv(m))/rtemp

```



```

      A(N,M)=(TOP/(rtemp))*(dxr*xn(n)+dyr*yn(n))
c      write(89,*)a(n,m)
c      The above include the influence of the vortex influence on
c      the foil itself. Now, add in the images above and below
c      the foils due to wall effects. Need Tunparam
c
      DO 710 K=1,NIMAGEPR
c
c      Test K to see if it is a positive or negative image line
c      if true then it is a positive image line
c      if not true then it is a negative image
c
      if ((2*int(K/2)).eq.k) then
        signimage=1.0
      else
        signimage=-1.0
      endif
c      DEBUGGER
c      if ((n.eq.1).and.(m.eq.1)) then
c        write(*,*) signimage
c      endif
c      First Add in the Effect due to the image ABOVE (in pos y-dir)
c
      rtemp=sqrt((xv(m)-xc(n))**2+
%      ((k*yimage+ signimage*yv(m))-yc(n))**2)
      dyr=-(xc(n)-xv(m))/rtemp
      dxr=-(yc(n)-(k*yimage+signimage*yv(m)))/rtemp
      A(N,M)= A(N,M)+signimage*(TOP/(rtemp))*(dxr*xn(n)+dyr*yn(n))
c
c      Next, Add in the Effect due to the image BELOW (in neg y-dir)
c
      rtemp=sqrt((xv(m)-xc(n))**2+
%      ((-k*yimage+ signimage*yv(m))-yc(n))**2)
      dyr=-(xc(n)-xv(m))/rtemp
      dxr=-(yc(n)-(-k*yimage+signimage*yv(m)))/rtemp
      A(N,M)= A(N,M)+signimage*(TOP/(rtemp))*(dxr*xn(n)+dyr*yn(n))

710      continue

c
c
c
700  CONTINUE
c
c      FACTOR THE A MATRIX
c
      CALL FACTOR(A,IPIVOT,WKAREA,NPAN,NI,IERR)
c
c      COMPUTE THE w VELOCITY AT THE NTH CONTROL POINT
c      THE B MATRIX OF A*GAMMA=B
c
      DO 800 N=1,NPAN
        B(N)=DYDX(N)
c        write(*,*)dydx(n)
800  CONTINUE
c
c      NOW BACK SUBTITUTE TO EXTRACT THE VORTEX STRENGTHS
c
      CALL SUBST(A,B,GAMMA,IPIVOT,NPAN,NI)
c
c      check the circulation distribution on the foil
c
c      write out to fort.89
c

```

```

sum=0.0
do 900 i=1,npan
    sum=sum+gamma(i)
    write(89,*)xv(i),gamma(i)/ds(i)
900 continue
write(*,*) 'Estimated CL=',sum*2.0
write(*,*) 'Using Vortex Lattice'

c
c    extract the wake
c
c    The circulation is now known for each vortex point.
c    To find the wake, "Step" off the trailing edge and
c    evaluate the field point velocity. March off a distance
c    in that direction. Evaluate the field point velocity again.
c    adjust course. Keep a record of the path. That will be
c    the line for the wake centerline.
c
c    The following quantities are needed:
c    NWAKE, DSL
c    Want to calculate: WAKEX,WAKEX @ NWAKE points between TE and DSL
c
c    March down the wake first, spline the values and then extract
c    the NWAKE points.
c
c    Where is the Trailing edge? @ pressx(1),pressy(y)
c
c    Where do you want to stop? @ pressx(1)+DSL & corresponding y
c
    endx=pressx(1)+dsl+1.0
c
c    WHAT STEP DO YOU WANT TO MARCH IN? (think of as scaled time)
c    STEP=DSL/25.0
c
c    START MARCHING
c
    I=1
    TX(1)=PRESSX(1)
    TY(1)=PRESSY(1)
    write(88,*)'zone'
    DO 910 WHILE (TX(I).LT.ENDX)
c
c    FIND FIELD POINT VELOCITY DISTURBANCE DUE TO VORTICES
c    REFERENCE NEWMAN PAGE 190
c    U=1 to add in the freestream effect
        U=1.0
        V=0.0
        DO 920 N=1,NPAN
            TEMP=2*PI*((XV(N)-TX(I))*(XV(N)-TX(I))+(YV(N)-TY(I))*
%              (YV(N)-TY(I)))
            U=U+GAMMA(N)*(TY(I)-YV(N))/TEMP
            V=V-GAMMA(N)*(TX(I)-XV(N))/TEMP
c
c    Add in the influence of the images
c
c
        DO 1950 K=1,NIMAGEPR
c
c    Test K to see if it is a positive or negative image line
c    if true then it is a positive image line
c    if not true then it is a negative image
c
            if ((2*int(K/2)).eq.k) then
                signimage=1.0
            else
                signimage=-1.0

```

```

        endif
c      DEBUGGER
c      First Add in the Effect due to the image ABOVE (in pos y-dir)
c
      temp=2*PI*(((XV(N)-TX(I))**2+
%          ((k*yimage+ signimage*yv(n))-ty(i))**2))
%
      U=U+signimage*GAMMA(N)*(TY(I)-(k*yimage+
%          signimage*yv(n)))/TEMP
      V=V-signimage*GAMMA(N)*(TX(I)-XV(N))/TEMP
c
c      Next, Add in the Effect due to the image BELOW (in neg y-dir)
c
      temp=2*PI*(((XV(N)-TX(I))**2+
%          ((-k*yimage+ signimage*yv(n))-ty(i))**2))
%
      U=U+signimage*GAMMA(N)*(TY(I)-(-k*yimage+
%          signimage*yv(n)))/TEMP
      V=V-signimage*GAMMA(N)*(TX(I)-XV(N))/TEMP

1950      continue

c
c
c
c
c
920      continue
c
c      CALCULATE THE NEXT LOCATION TO LOOK
c
c      Make the first step a baby step
c
      IF(I.LE.3)THEN
      STEP=STEP/8.0
      ELSE
      STEP=DSL/25.0
      ENDIF

c
c      Where is the next wake location?
c
      TX(I+1)=TX(I)+U*STEP
      TY(I+1)=TY(I)+V*STEP
      write(88,*)tx(i),ty(i),0,0
      I=I+1
910      END DO
      NTOT=I

c
c      SPLINE THE POINTS TO OBTAIN CUBIC COEFFICIENTS
c
c...SPLINE SPACING WITH FIXED SLOPE AT THE ENDS
      CALL UGLYDK(NTOT,1,1,TX,TY,0,0,WAKCUB)
c...EVALUATE SPLINE TO FIND STEP SIZE AT INTERMEDIATE POINTS
c
c      WHERE ARE THE OUTPUT POINTS?
c
      WAKEX(1)=PRESSX(1)
      DO 950 I=1,NWAKE-1
      WAKEX(I+1)=WAKEX(I)+DSL/FLOAT((NWAKE-1))
950      CONTINUE
c
c      EXTRACT THE POINTS FOR THE WAKEX AND WAKY TO RETURN TO MAIN

```

```

C
      CALL EVALDK(NTOT,NWAKE,TX,WAKEX,WAKEY,WAKCUB)
      write(88,*)'ZONE T=WAKEADS'
      DO 960 I=1,10
      WRITE(88,*)WAKEX(I),WAKEY(I),0,0
960    CONTINUE

      write(*,*) '*****'
      write(*,*) 'THE GRID BOUNDARIES HAVE BEEN ADAPTED TO THE WAKE'
      write(*,*) '          USING VORTEX LATTICE METHOD'
      write(*,*) '*****'
      RETURN
      END

C
C+++++
*****
*   SINGLE PRECISION VERSION OF DAVE GREELEY'S DIRECT SOLVER   *
*****
C2345678901234567890123456789012345678901234567890123456789012
C.....Subroutine FACTOR.....
      SUBROUTINE FACTOR(W,IPIVOT,D,N,NDIM,IERR)
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION W(NDIM,NDIM),IPIVOT(*),D(*)
      IERR=0
      DO 10 I=1,N
        IPIVOT(I)=I
        ROWMAX=0.
        DO 9 J=1,N
          ROWMAX=MAX(ROWMAX,ABS(W(I,J)))
9        CONTINUE
        IF(ROWMAX.EQ.0.) GO TO 999
        D(I)=ROWMAX
10       CONTINUE
        NM1=N-1
        IF(NM1.EQ.0) RETURN
        DO 20 K=1,NM1
          J=K
          KP1=K+1
          IP=IPIVOT(K)
          COLMAX=ABS(W(IP,K))/D(IP)
          DO 11 I=KP1,N
            IP=IPIVOT(I)
            AWIKOV=ABS(W(IP,K))/D(IP)
            IF(AWIKOV.LE.COLMAX) GO TO 11
            COLMAX=AWIKOV
            J=I
11          CONTINUE
          IF(COLMAX.EQ.0.) GO TO 999
          IPK=IPIVOT(J)
          IPIVOT(J)=IPIVOT(K)
          IPIVOT(K)=IPK
          DO 20 I=KP1,N
            IP=IPIVOT(I)
            W(IP,K)=W(IP,K)/W(IPK,K)
            RATIO=-W(IP,K)
            DO 20 J=KP1,N
              W(IP,J)=RATIO*W(IPK,J)+W(IP,J)
20          CONTINUE
          IF(W(IP,N).EQ.0.) GO TO 999
          RETURN
999      IERR=2
          RETURN
          END
C.....End of FACTOR.....

```

```

C.....Subroutine SUBST.....
C*****

      SUBROUTINE SUBST(W,B,X,IPIVOT,N,NDIM)
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION W(NDIM,NDIM),B(*),X(*),IPIVOT(*)
      IF(N.GT.1) GO TO 10
      X(1)=B(1)/W(1,1)
      RETURN
10    IP=IPIVOT(1)
      X(1)=B(IP)
      DO 15 K=2,N
        IP=IPIVOT(K)
        KM1=K-1
        SUM=0.
        DO 14 J=1,KM1
          SUM=W(IP,J)*X(J)+SUM
14    CONTINUE
        X(K)=B(IP)-SUM
15    CONTINUE
      X(N)=X(N)/W(IP,N)
      K=N
      DO 20 NP1MK=2,N
        KP1=K
        K=K-1
        IP=IPIVOT(K)
        SUM=0.0
        DO 19 J=KP1,N
          SUM=W(IP,J)*X(J)+SUM
19    CONTINUE
        X(K)=(X(K)-SUM)/W(IP,K)
20    CONTINUE
      RETURN
      END

C.....End of SUBST.....
C*****
*****
*END OF SINGLE PRECISION VERSION OF DAVE GREELEY'S DIRECT SOLVER *
*****

C
C+++++
C
      SUBROUTINE MAKEDUM(AX,AY,IMAXA,JMAXA,BX,BY,IMAXB,JMAXB,
      +XOUT,YOUT,IMAXOUT,JMAXOUT)
C
C
C      This subroutine takes in two axially adjacent arrays and
c      appends the imaxa-1 vertical line of points to the left
c      hand side of the downstream array.
c
c      Last Updated:11 November
c
      IMPLICIT NONE
      INTEGER IMAXA,JMAXA,IMAXB,JMAXB,IMAXOUT,JMAXOUT
      INTEGER I,NI,J
      PARAMETER(NI=200)
C
C      ARRAYS
C
      REAL AX(NI,NI),AY(NI,NI),BX(NI,NI),BY(NI,NI)
      REAL XOUT(NI,NI),YOUT(NI,NI)
C
C

```

```

      DO 100 J=1,JMAXA
        XOUT(1,J)=AX(IMAXA-1,J)
        YOUT(1,J)=AY(IMAXA-1,J)
100    CONTINUE

      DO 200 I=1,IMAXB
        DO 300 J=1,JMAXB
          XOUT(I+1,J)=BX(I,J)
          YOUT(I+1,J)=BY(I,J)
300    CONTINUE
200    CONTINUE
      IMAXOUT=IMAXB+1
      JMAXOUT=JMAXB
      RETURN
      END
C+++++
      SUBROUTINE ZONELINES(X,Y,IMAX,JMAX,NLINE,NFILE)
C
C   This subroutine assists in writing output file for FIT2D
c   to INMESH output file.
c
c   X,Y are arrays containing points
c   IMAX,JMAX are # of array elements
c   NLINE is the line number in the zone
c   NFIEL is the file output number
C
      IMPLICIT NONE
      INTEGER NI,NJ,I, IMAX,JMAX,J,NLINE,NFILE
      PARAMETER (NI=200,NJ=200)
      REAL X(NI,NJ),Y(NI,NJ)
      REAL X1(NI,NJ),X2(NI,NJ),Y1(NI,NJ),Y2(NI,NJ)
      REAL RI1,RI2,RJ1,RJ2,BETA
C
C
C
      IF(NLINE.EQ.1)THEN
        DO 100 J=1,JMAX
          WRITE(NFILE,*)X(1,J),Y(1,J)
100    CONTINUE
        ENDIF
      IF(NLINE.EQ.3)THEN
        DO 300 J=1,JMAX
          WRITE(NFILE,*)X(IMAX,J),Y(IMAX,J)
300    CONTINUE
        ENDIF
      IF(NLINE.EQ.2)THEN
        DO 200 I=1,IMAX
          WRITE(NFILE,*)X(I,1),Y(I,1)
200    CONTINUE
        ENDIF
      IF(NLINE.EQ.4)THEN
        DO 400 I=1,IMAX
          WRITE(NFILE,*)X(I,JMAX),Y(I,JMAX)
400    CONTINUE
        ENDIF

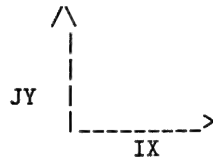
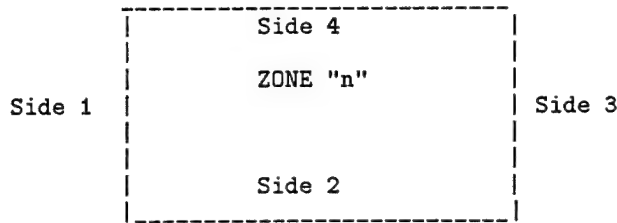
      RETURN
      END
C
C+++++
C
      SUBROUTINE BORDERS(X,Y,X1,Y1,X2,Y2,X3,Y3,X4,Y4,IX,JY,NI)
C
C   This subroutine takes in the x,y points which represent the four

```

```

c lines surrounding a 2d box. It then writes them to a 2d array
c which represents the box boundaries. The "interior" of the 2d
c array will still be zeros. The interior is filled in another
c subroutine which interpolates between all the points.

```



Array reference to enter finite interpolation scheme.

```

c
c IMPLICIT NONE
c INTEGER I,IX,JY,NTEMP,NI
c
c ARRAYS
c
c REAL X(NI,NI),Y(NI,NI),X1(NI),Y1(NI),X2(NI),Y2(NI)
c REAL X3(NI),Y3(NI),X4(NI),Y4(NI)
c REAL X5(NI),Y5(NI),X6(NI),Y6(NI)
c
c Check line One: Determine if the order forward up or forward down.
c
c ntemp = 0 then count forward
c ntemp = 1 count backwards
c
c debugger
c
c write(*,*)'Debugging in borders y(jmax), y(1)'
c write(*,*) y1(jy),y1(1)
c NTEMP=0
c IF (Y1(JY).LT.Y1(1))THEN
c   NTEMP=1
c ENDIF
c
c Now put line one in to x and y arrays
c
c DO 100 I=1,JY
c   IF (NTEMP.EQ.0) THEN
c     X(1,I)=X1(I)
c     Y(1,I)=Y1(I)
c   ELSE
c     X(1,I)=X1(JY-I+1)
c     Y(1,I)=Y1(JY-I+1)
c   ENDIF
c
c debugger
c
c if (i.eq.1)then
c   write(80,*) "ZONE"
c endif
c write(80,*) x(1,I),y(1,I)
c

```

```

c      end debugger
c
100  CONTINUE
c
c      Check line Two: Determine if the order is forward or backwards
c
c      ntemp = 0 then count forward
c      ntemp = 1 count backwards
c      NTEMP=0
c      IF (X2(IX).LT.X2(1))THEN
c          NTEMP=1
c      ENDIF
c
c      Now put line two in to x and y arrays
c
c      DO 200 I=1,IX
c          IF (NTEMP.EQ.0) THEN
c              X(I,1)=X2(I)
c              Y(I,1)=Y2(I)
c          ELSE
c              X(I,1)=X2(IX-I+1)
c              Y(I,1)=Y2(IX-I+1)
c          ENDIF
c
c      debugger
c
c      if (i.eq.1)then
c          write(80,*) "ZONE"
c      endif
c      write(80,*) x(I,1),y(I,1)
c
c      end debugger
c
200  CONTINUE
c
c      Check line Three:Determine if the order forward up or forward down.
c
c      ntemp = 0 then count forward
c      ntemp = 1 count backwards
c
c      NTEMP=0
c      IF (Y3(JY).LT.Y3(1))THEN
c          NTEMP=1
c      ENDIF
c
c      Now put line three in to x and y arrays
c
c      DO 300 I=1,JY
c          IF (NTEMP.EQ.0) THEN
c              X(ix,I)=X3(I)
c              Y(ix,I)=Y3(I)
c          ELSE
c              X(ix,I)=X3(JY-I+1)
c              Y(ix,I)=Y3(JY-I+1)
c          ENDIF
c
c      debugger
c
c      if (i.eq.1)then
c          write(80,*) "ZONE"
c      endif
c      write(80,*) x(ix,I),y(ix,I)
c
c      end debugger
c

```



```

300  CONTINUE
C
C****
C
C
C    Check line Four:  Determine if the order is forward or backwards
C
C    ntemp = 0 then count forward
C    ntemp = 1 count backwards
C    NTEMP=0
C    IF (X4(IX).LT.X4(1))THEN
C        NTEMP=1
C    ENDIF
C
C    Now put line four in to x and y arrays
C
C    DO 400 I=1,IX
C        IF (NTEMP.EQ.0) THEN
C            X(I,jy)=X4(I)
C            Y(I,jy)=Y4(I)
C        ELSE
C            X(I,jy)=X4(IX-I+1)
C            Y(I,jy)=Y4(IX-I+1)
C        ENDIF
C
C    debugger
C
C    if (i.eq.1)then
C        write(80,*) "ZONE"
C    endif
C    write(80,*) x(I,jy),y(I,jy)
C
C    end debugger
C
400  CONTINUE
C
    CALL interp(X,Y,IX,JY)
    RETURN
    END
C+++++
C
    SUBROUTINE interp(X,Y,IMAX,JMAX)
C
C
C    Below is a subroutine for doing ISOPARAMETRIC interpolation
C    for a single zone.  It assumes that all four edges are
C    already split up to their desired spacings.
C    This method is based on the simple isoparemetric method presented
C    in "FINITE ELEMENT PROCEDURES" by Bathe.
C
C
C    Written by John Dannecker for use in FIT2D.F
C    Last Modified:  November 7, 1996
C
C
C    IMPLICIT NONE
C    INTEGER NI,NJ,I, IMAX,JMAX,J
C    PARAMETER (NI=200,NJ=200)
C    REAL X(NI,NJ),Y(NI,NJ)
C    REAL X1(NI,NJ),X2(NI,NJ),Y1(NI,NJ),Y2(NI,NJ)
C    REAL RI1,RI2,RJ1,RJ2,beta
C
C    i=15
C    j=25
C    DO ALL THE X(I,J)

```

```

C
DO 100 I=2,IMAX-1
  RI1=(X(I,1)-X(1,1))/(X(IMAX,1)-X(1,1))
  ri2=(x(i,jmax)-x(1,jmax))/(x(imax,jmax)-x(1,jmax))
C
  RI1=RI1+(X(I,JMAX)-X(1,jmax))/(X(IMAX,JMAX)-X(1,JMAX))
C
DO 200 J=2,JMAX-1
  beta=(Y(1,J)-Y(1,1))/(Y(1,JMAX)-Y(1,1))
  RJ1=RJ1+(Y(IMAX,J)-Y(IMAX,1))/(Y(IMAX,Jmax)-Y(IMAX,1))
C
  X(I,J)=x(1,j)+(x(imax,j)-x(1,j))*((ri1*(1.0-beta))+(ri2*(beta)))
200  CONTINUE
100  CONTINUE
C
C DO ALL THE Y(I,J)
C
DO 300 j=2,jMAX-1
  Rj1=(y(1,j)-y(1,1))/(y(1,jmax)-y(1,1))
  Rj2=(y(imax,j)-y(imax,1))/(y(IMAX,JMAX)-y(imax,1))
C
DO 400 i=2,iMAX-1
  beta=(x(i,1)-x(1,1))/(x(imax,1)-x(1,1))
  Ri2=Ri2+(x(i,jmax)-x(1,jmax))/(x(IMAX,jmax)-x(1,jmax))
C
  Y(I,J)=y(i,1)+(y(i,jmax)-y(i,1))*(rj1*(1.0-beta)+rj2*beta)
400  CONTINUE
300  CONTINUE
RETURN
END
C
C
C+++++
C
SUBROUTINE FINDYPLUS(RESBL,RESWALL,REYNOLD,TUNPARAM,USL,DSL)
C
C This subroutine calculates the parameter y+
C using the method in Anderson,Tannehill,Pletcher
C Computaional Fluid Mechanics and Heat Transfer, 1984.
C
C Y+ is checked against the user input values for cell height
C on the foil and wall. If the user cell spacing exceeds
C y+ criteria, then cell spacing can be changed.
C IMPLICIT NONE
C REAL RESBL,RESWALL,REYNOLD,TUNPARAM,USL,DSL,HEIGHT,NU
C REAL LF,LW, VEL, YPLUSF,REYWALL,YPLUSW,TEMP
C CHARACTER ANSWER*3
C
C MIT WATERTUNNEL CROSS SECTION HEIGHT (FEET)
C
C HEIGHT=20.0/12.0
C
C KINEMATIC VISCOSITY (NU FT^2/SEC)
C FRESH WATER @ 70 DEGREES F.(SOURCE PNA 1967)
C NU=1.0519E-5
C
C CALCULATE CHORD LENGTH
C
C LF=TUNPARAM*HEIGHT
C
C Calculate wall length
C
C LW=TUNPARAM*HEIGHT*(1.0+USL+DSL)
C

```

```

C      CALCULATE FLOW SPEED IN TUNNER (FT/SEC)
C
C      VEL=NU*REYNOLD/LF
C
C      CALCULATE YPLUS FOR FOIL
C
C      YPLUSF=(3.0/(REYNOLD**(-0.10)*SQRT(0.0227)*VEL/NU))/LF
C
C      CALCULATE WALL BASED REYNOLD NUMBER
C
C      REYWALL=VEL*LW/NU
C
C      CALCULATE YPLUS AT WALL
C
C      YPLUSW=(3.0/(REYWALL**(-0.10)*SQRT(0.0227)*VEL/NU))/LF
C
C      COMPARE USER INPUT CELL HEIGHT TO Y+:
C
C      debugger
C
99      FORMAT(A12,F10.6,A6,F10.6)
      write(*,*) 'Cell Dim   User Input Val   Yplus(3)'
      write(*,99) 'On Foil: ',resbl,' ',yplusf
      write(*,99) 'On Wall: ',reswall,' ',yplusw
C
C      ON THE FOIL:
      TEMP=YPLUSF/RESBL
      IF (RESBL.GE.YPLUSF) THEN
        WRITE(*,*) 'USER INPUT CELL HEIGHT ON FOIL IS GREATER THAN'
        WRITE(*,*) 'CALCULATED Y+(3) VALUE.'
        WRITE(*,*(A,$)) 'DO YOU WANT TO CHANGE TO CALC Y+(3)?<n>:'
        READ(*,*(A)) ANSWER
        IF ((ANSWER.EQ.'Y').OR.(ANSWER.EQ.'y')) THEN
          RESBL=YPLUSF
          WRITE(*,*) 'CELL HEIGHT ON FOIL SET TO Y+(3).'
        ELSE
          WRITE(*,*) 'CELL HEIGHT ON FOIL IS UNCHANGED.'
        ENDIF
      ELSE
        WRITE(*,*) 'USER SPECIFIED CELL HEIGHT ON FOIL IS ADEQUATE'
      ENDIF
C
C      AT THE WALL
      TEMP=YPLUSW/RESWALL
      IF (RESWALL.GE.YPLUSW) THEN
        WRITE(*,*) 'USER INPUT CELL HEIGHT AT WALL IS GREATER THAN'
        WRITE(*,*) 'CALCULATED Y+(3) VALUE.'
        WRITE(*,*(A,$)) 'DO YOU WANT TO CHANGE TO CALC Y+(3)?<n>:'
        READ(*,*(A)) ANSWER
        IF ((ANSWER.EQ.'Y').OR.(ANSWER.EQ.'y')) THEN
          RESWALL=YPLUSW
          WRITE(*,*) 'CELL HEIGHT AT WALL SET TO Y+(3).'
        ELSE
          WRITE(*,*) 'CELL HEIGHT AT WALL IS UNCHANGED.'
        ENDIF
      ELSE
        WRITE(*,*) 'USER SPECIFIED CELL HEIGHT AT WALL IS ADEQUATE'
      ENDIF
C
C      RETURN
      END
C
C+++++

```

```

SUBROUTINE TUND(XBEG,YBEG,XEND,YEND,RESBL,RESWALL,XOUT,YOUT,
+NPOINTS,PACKING)
C
C   LAST MODIFIED:29 OCTOBER 1996
C
C   This subroutine generates the vertical spacing of points on
c   lines above and below the foil.
c
C   IMPLICIT NONE
REAL PI,TWOPI,ZERO,ONE,HALF
INTEGER NI,I,NNEARBL,NNEARWALL,NREMAIN
PARAMETER (PI=3.14159,TWOPI=6.28318, NI=200, ZERO=0.0, ONE=1.0)
INTEGER NPOINTS
REAL XBEG,XEND,YBEG,YEND,RESBL,RESWALL,PACKING
REAL DELX,DELY,TEMPX,TEMPY,DLEFT,DRIGHT
real xvect,yvect,hypo
C
C   ARRAYS
REAL XOUT(NI),YOUT(NI),XLIN(NI),YLIN(NI),XREM(NI),YREM(NI)
C
C   Determine spacing of points for boundary layer along foil
c
c   How many points close packed near foil and at the wall?
c
NNEARBL=INT(PACKING*FLOAT(NPOINTS))
NNEARWALL=NNEARBL
NREMAIN=NPOINTS-NNEARWALL-NNEARBL+2
C
C   Size the cells on the foil:
c
XOUT(1)=XBEG
YOUT(1)=YBEG
DELX=XEND-XBEG
DELY=YEND-YBEG
hypo=sqrt(delx*delx+dely*dely)
xvect=delx/hypo
yvect=dely/hypo
c
c   debugger
c   write(*,*)xvect,yvect,delx,xend,xbeg
C
C   Each cell is 25% larger than the previous cell.
c
DO 100 I=1,NNEARBL-1
  XOUT(I+1)=XOUT(I)+(1.25**(I-1))*RESBL*xvect
  YOUT(I+1)=YOUT(I)+(1.25**(I-1))*RESBL*yvect
c   write(*,*)i+1,XOUT(I+1),YOUT(I+1)
100 CONTINUE
DLEFT=(1.25**(I-1))*RESBL
C
C   Size the cells at the wall:
c
XOUT(NPOINTS)=XEND
YOUT(NPOINTS)=YEND
C
C   NOTE: This is a REVERSE counter!!!!
c
DO 200 I=1,NNEARWALL-1
C
C   Each cell is 25% larger than the previous cell.
c
XOUT(NPOINTS-I)=XOUT(NPOINTS-I+1)-(1.25**(I-1))*RESWALL*xvect
YOUT(NPOINTS-I)=YOUT(NPOINTS-I+1)-(1.25**(I-1))*RESWALL*yvect
c   write(*,*)npoints-i,XOUT(NPOINTS-I),YOUT(NPOINTS-I)
200 CONTINUE
DRIGHT=(1.25**(I-1))*RESWALL

```

```

C
C   Send the remainder of the line to FNSPLT:
C
C
C   First need to generate a small line
C
C       Endpoints of line: XOUT(NNEARBL),YOUT(NNEARBL)
C                           XOUT(NPOINTS-NNEARWALL),YOUT(NPOINTS-NNEARWALL)
C   FNSPLT WILL NEED 5 POINTS
C   TEMPX=XOUT(NPOINTS-NNEARWALL+1)-XOUT(NNEARBL)
C   TEMPY=YOUT(NPOINTS-NNEARWALL+1)-YOUT(NNEARBL)
C
C   debugger
C   write(*,*)tempX,tempY,XOUT(NPOINTS-NNEARWALL)
C
C   XLIN(1)=XOUT(NNEARBL)
C   YLIN(1)=YOUT(NNEARBL)
C   DO 300 I=1,4
C       XLIN(I+1)=XLIN(I)+0.25*TEMPX
C       YLIN(I+1)=YLIN(I)+0.25*TEMPY
C       write(*,*)'Straight Line'
C       write(*,*)xlin(i),ylin(i)
300  CONTINUE
C
C
C   CALL FNSPLT(5,NREMAIN,DLEFT,DRIGHT,XLIN,YLIN,XREM,YREM)
C
C   FILL OUT THE ARRAY
C   DO 400 I=1,NREMAIN
C       XOUT(NNEARBL+I-1)=XREM(I)
C       YOUT(NNEARBL+I-1)=YREM(I)
400  CONTINUE
C
C   debugger
C   WRITE(*,*)NNEARBL,NNEARWALL,NPOINTS,NREMAIN
C
C   debugger
C
C
C   write(79,*) 'ZONE'
C   do 500 i=1,npoints
C       write(79,*) xout(i),yout(i)
c 500  continue
C
C   end debugger
C
C
C   RETURN
C   END
C+++++
C   SUBROUTINE TUNC(XBEG,XEND,YWALL,NPOINTS,XIN,YIN,XOUT,YOUT)
C
C   LAST MODIFIED:08 NOVEMBER 1996
C
C   This subroutine takes the cell spacing along the upstream line
C   in the center of the tunnel, the foil points and the down
C   stream line and spaces them along tunnel walls.
C
C   XBEG = BEGINNING XCOORD OF LINE
C
C   XEND = ENDING XCOORD OF LINE
C
C   YWALL = Y VALUE AT WALL

```

```

C
C      NPOINTS = NUMBER OF ARRAY VALUES COMING IN
C
C      XIN,YIN, = VALUES COMING IN THAT WILL BE PROJECTED
C
C      XOUT,YOUT= X AND Y VALUES FOR THE LINE ON THE WALL
C
      IMPLICIT NONE
      REAL PI,TWOPI,ZERO,ONE,HALF
      INTEGER NI,I
      PARAMETER (PI=3.14159,TWOPI=6.28318, NI=200, ZERO=0.0, ONE=1.0)
      INTEGER NPOINTS
      REAL XEND, XBEG,DELX,DELY,TEMP3,SLENGTH,TEMP4,YWALL
      REAL XIN(NI), YIN(NI), XOUT(NI), YOUT(NI),DELARC(NI)
C
C      Compute arc length of the input array and arc length between
c      points.
c
      SLENGTH=ZERO
      DELX=ZERO
      DELY=ZERO
      TEMP3=ZERO
      DO 100 I=1,NPOINTS-1
        DELX=XIN(I+1)-XIN(I)
        DELY=YIN(I+1)-YIN(I)
        TEMP3=DELX*DELX+DELY*DELY
        DELARC(I)=SQRT(TEMP3)
        SLENGTH=SLENGTH+DELARC(I)
100    CONTINUE
C
C      debugger
C      temp4=xin(npoints)-xin(1)
C      write(*,*)slength,temp4,ywall,XBEG,XEND
c
c      do algebraic translation of point spacing
c
      TEMP4=ZERO
      XOUT(1)=XBEG
      YOUT(1)=YWALL
      DO 200 I=1,NPOINTS-1
        TEMP4=DELARC(I)/SLENGTH
        XOUT(I+1)=XOUT(I)+TEMP4*(XEND-XBEG)
        YOUT(I+1)=YWALL
200    CONTINUE
C
C      debugger
C
c      write(79,*) 'ZONE'
c      do 300 i=1,npoints
c        write(79,*) xout(i),yout(i)
c 300    continue
c
c      end debugger
c
      RETURN
      END
C+++++
      SUBROUTINE WALLFIND(SCALE,YUP,YBOT)
C
C      LAST MODIFIED:  29 OCTOBER 96
C
C      This subroutine finds the upper and lower y coordinates of the
c      tunnel walls
c

```

```

C
  IMPLICIT NONE
  REAL PI,TWOPI,ZERO,ONE,HALF,TWO
  INTEGER NI
  PARAMETER (PI=3.14159,TWOPI=6.28318,NI=10,
+ZERO=0.0,ONE=1.0,TWO=2.0)
  REAL SCALE,YUP,YBOT
C
  YUP=ONE/(SCALE*TWO)
  YBOT=-ONE/(SCALE*TWO)
C
  debugger
C
  write(*,*) yup,ybot
C
C
  RETURN
  END
C+++++
  SUBROUTINE TUNB(XL,XLOC,YLOC,PLINEX,PLINEY,NP)
C
C   LAST MODIFIED:   24 OCTOBER 96
C
C   This subroutine defines the upstream or downstream lines extending
c   from the foil leading or trailing edge.
c
C   XL = LENGTH OF LINE
C
C   XLOC,YLOC = BEGINNING OF LINE
C
C   PLINEX,PLINEY = OUTPUT X AND Y
C
C   NP = NUMBER OF POINTS OUT
  IMPLICIT NONE
  REAL PI,TWOPI,ZERO,ONE,HALF
  INTEGER NP
  PARAMETER (PI=3.14159,TWOPI=6.28318, ZERO=0.0, ONE=1.0)
  REAL XL,XLOC,YLOC,TEMP
  INTEGER I
  REAL PLINEX(NP),PLINEY(NP)
C
C   debugger
c   write(*,*) 'Made it in to TUNB, LIMIT IS:', XL
c
C   MAKE IT HORIZONTAL
  DO 100 I=1,NP
    PLINEY(I)=YLOC
100  CONTINUE
C
C   debugger
c   write(*,*) xloc,yloc
C
C   The endpoints of the line are xloc,yloc and xloc+temp,yloc
  IF (XLOC.LE.ZERO)THEN
    PLINEX(NP)=XLOC-XL
    PLINEX(1)=XLOC
  ENDIF
C
  IF (XLOC.GT.ZERO)THEN
    PLINEX(NP)=XLOC+XL
    PLINEX(1)=XLOC
  ENDIF
C
C   Generate some points between the two endpoints.
c

```

```

        TEMP=(PLINEX(NP)-PLINEX(1))/FLOAT(NP-1)
        DO 200 I=1,NP-2
            PLINEX(I+1)=PLINEX(I)+TEMP
200    CONTINUE

C
C    DEBUGGER
c    write(79,*) 'ZONE'
c    do 5020 i=1,NP
c        write(79,*) PLINEX(i),PLINEY(i)
c 5020 continue
        RETURN
        END

C+++++
      SUBROUTINE ARCJOINER(N1,N2,NOUT,X1,X2,Y1,Y2,XOUT,YOUT)
C
C    last modified: 25 October 96
C
C    This subroutine takes in two arcs and joins them end to end
c    in to one array.
      IMPLICIT NONE
      REAL PI,TWOPI,ZERO,ONE,HALF
      INTEGER NI,I
      PARAMETER (PI=3.14159,TWOPI=6.28318, NI=200, ZERO=0.0, ONE=1.0)
      INTEGER N1, N2,NOUT
      REAL X1(NI), Y1(NI), XOUT(NI), YOUT(NI), X2(NI),Y2(NI)
C
C    debugger
c    write(*,*) 'Made it in to ARCJOINER:', n1,n2
c
c
      DO 10 I=1,N1
          XOUT(I)=X1(I)
          YOUT(I)=Y1(I)
10    CONTINUE
      DO 20 I=2,N2
          XOUT(I+N1-1)=X2(I)
          YOUT(I+N1-1)=Y2(I)
20    CONTINUE
C
c    funny counting here eliminates overlapping data points
c
      NOUT=N1+N2-1
C
C    debugger
c    write(*,*) 'Points Out',nout
c
c
      RETURN
      END

C+++++
      SUBROUTINE ARCMAKER(X,Y,NUM1,NUM2,XO,YO)
C
C    last modified: 29Sept96
C
C    THIS SUBROUTINE TAKES IN LONG ARRAYS X, AND Y,
C    AND RETURNS PORTIONS OF X AND Y SPECIFIED BY
C    INTEGERS NUM1 AND NUM2. XO AND YO ARE FILLED
C    BY THE FIRST NUM1-NUM2 ELEMENTS
C
      IMPLICIT NONE

```



```

C
C  debugger
C  write(*,*) 'pressmid=', chordmid
C
DO 20 I=1,(NOSE-1)
  IF (XI(I).GT.CHORDMID) THEN
    TEMP=I
  ENDIF
20 CONTINUE
MIDPRES=TEMP
C
C  FIND THE MIDDLE OF THE SUCTION SIDE
C
TEMP=0
CHORDMID=(XI(NOSE)+XI(NPOINTS))/(2.0)
C  write(*,*) 'suctmid=', chordmid
DO 30 I=NOSE,(NPOINTS-1)
  IF (XI(I).LT.CHORDMID) THEN
    TEMP=I
  ENDIF
30 CONTINUE
MIDSUCT=TEMP
C
C  debugger
C
C  WRITE(*,*) NPOINTS,NOSE,MIDSUCT,MIDPRES
C
RETURN
END
C
C+++++
SUBROUTINE NACACONV(N,X,Y,T,R)
C
C  last modified: 23 Sept 96
C
C  THIS SUBROUTINE CONVERTS NACA FOIL GEOMETRY TO X,Y GEOMETRY
C
C  N = NUMBER OF STATIONS
C  X = STATION
C  Y = CAMBER
C  T = THICKNESS
C  R = L.E. RADIUS
C
C  RETURNS
C  N = NUMBER OF POINTS (.GT. 2*STATIONS)
C  X,Y = GEOMETRY COORDINATES
C
  IMPLICIT NONE
  REAL PI,TWOPI,ZERO,ONE,HALF
  INTEGER NI
  PARAMETER(PI=3.14159,TWOPI=6.28318, NI=200)
  INTEGER I,N
  REAL R
  REAL X(NI),Y(NI),T(NI)
C
C  THIS SUBROUTINE NOT YET FULLY IMPLEMENTED
C
C  WRITE(*,*) ' THE NACA CONVERSION ROUTINE IS NOT YET IMPLEMENTED'
C  WRITE(*,*) ' Pretty lame, huh?'
C  END
C
C+++++
SUBROUTINE ROTANGLE(N,X,Y,AOA,PIVOTX,PIVOTY)
C
C  last modified: 26 Sept 96

```

```

C
C   THIS SUBROUTINE TAKES X,Y FOIL DATAPOINTS AND ROTATES THEM ABOUT
C   A PIVOT POINT TO SET FOIL TO PRESCRIBED ANGLE OF ATTACK(AOA)
C
C   N = NUMBER OF POINTS DEFINING FOIL
C   X = X COORDINATE OF POINT (X/C)
C   Y = Y COORDINATE OF POINT (Y/C)
C   AOA = FOIN ANGLE OF ATTACK IN DEGREES REF TO NOSE TAIL LINE
C   PIVOT = X/C OF ROTATION POINT OF FOIL
C
C   XL,YL ARE REF TO PIVOT POINT
C   R IS RADIUS TO PIVOT POINT
C
C   IMPLICIT NONE
C   REAL PI,TWOPI,ZERO,ONE,HALF
C   INTEGER NI
C   PARAMETER(PI=3.14159,TWOPI=6.28318, NI=200)
C   INTEGER I,N
C   REAL AOA, PIVOTX, PIVOTY, XL, YL, R, BETA
C   REAL X(NI),Y(NI)
C
C
C   WRITE(*,*) ' FOIL BEING ROTATED TO ANGLE OF ATTACK:' , AOA
C   WRITE(*,*) ' PIVOT POINT IS X COORD @ (X/C):' , PIVOTX
C   WRITE(*,*) ' PIVOT POINT IS Y COORD @ (Y/C):' , PIVOTY
C
C
C   ROTATE FOIL POINT BY POINT
C
C   DO 101 I =1,N
C       XL=X(I)-PIVOTX
C       YL=Y(I)-PIVOTY
C       R = SQRT(XL*XL+YL*YL)
C       BETA = ATAN2D(YL,XL)
C       X(I) = R*COSD(BETA-AOA)
C       Y(I) = R*SIND(BETA-AOA)
101  CONTINUE
C   RETURN
C   END
C
C+++++
C
C   SUBROUTINE NORMFOIL(L,N,X,Y)
C
C   LAST MODIFIED: 24 OCTOBER 96
C   IMPLICIT NONE
C   REAL PI,TWOPI,ZERO,ONE,HALF
C   INTEGER NI,I
C   PARAMETER (PI=3.14159,TWOPI=6.28318, NI=200, ZERO=0.0, ONE=1.0)
C   INTEGER N
C   REAL L
C   REAL X(NI),Y(NI)
C   DO 10 I=1,N
C       X(I)=X(I)/L
C       Y(I)=Y(I)/L
10  CONTINUE
C   RETURN
C   END
C
C
C+++++
C   SUBROUTINE FNSPLT(NIN,NOUT,DS1,DS2,XI,YI,XO,YO)
C
C   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
C   !   THIS PROGRAM CONVERTED IN TO A SUBROUTINE IN "FIT2D.F"   !

```

```

C      !   BY JOHN DANNECKER.  THIS PROGRAM IS ORIGINALLY WRITTEN      !
C      !   BY S.D. BLACK, MIT, WITH OTHER CREDITS AS INDICATED        !
C      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
C
C FANCY SPLITTING PROGRAM FOR DIVIDING UP A CURVE
C THERE ARE TWO METHODS THIS PROGRAM CAN BE RUN
C
C INPUT 1:
C   DATA SET (X,Y) FOR THE CURVE
C   DS1 - INTERVAL AT LEFT HAND END POINT
C   DS2 - RIGHT HAND END POINT INTERVAL
C   SET NOUT < 0
C
C INPUT 2:
C   SAME AS INPUT 1 BUT WITH THE NUMBER OF POINTS SPECIFIED
C
C OUTPUT
C   X,Y OF SPLIT UP CURVE BETWEEN THESE INTERVALS
C
C PURPOSE
C   USEFUL FOR CUSTOM GRIDING OF CURVES
C
C THE SLOPE OF THE INTERVAL DS# IS HELD TO BE ZERO AT THE END POINTS
C THE PROGRAM ASSUMES THE ENDS OF THE CURVE ARE AT THE ENDS OF THE INPUT
C
C WRITTEN 1/8/95 S. BLACK, MOD BY J. DANNECKER 9/28/96
C
C   This is the NEW and Improved FNSPLT modified by
C   S. Black 25 October 1996
C
C   PARAMETER (NI=200)
C   REAL XI(NI),YI(NI),XO(NI),YO(NI),CUB(4*(NI-1))
C   REAL SI(NI),CUB1(4*(NI-1)),CUB2(4*(NI-1))
C   REAL A(3),B(3),C(NI),D(NI),E(NI)
C   CHARACTER FIN*20
C   CHARACTER PROMPT2*34
C
C
C   debugger
C   write(*,*)'made it in to fnsplt', nin,nout
C#####
C
C CALCULATE LENGTH OF CURVE BASED ON INPUT POINTS
C   SL=0.0
C   DO 10 I=2,NIN
C     SL=SL+SQRT((XI(I)-XI(I-1))**2+(YI(I)-YI(I-1))**2)
10  CONTINUE
C CALCULATE THE NUMBER OF POINTS REQ'D IF NOT SPECIFIED
C   IF (NOUT.LE.0) THEN
C     DSAVG=(DS1+DS2)/2.0
C     NOUT=INT(SL/DSAVG)+1
C   END IF
C SPLINE INPUT ARRAY PARAMETRICALLY (BOTH X AND Y)
C   Array SI is returned with arclength parameters non-dimed from 0..1
C   CALL PUGLYDK(NIN,XI,YI,SI,CUB1,CUB2)
C SET UP ARRAYS CONTAINING SPACING
C   Array A Contains pointers for the first, middle and last
C   steps.
C   Array B contains the dS values at the two ends and a guess
C   at what the step size in the middle should be.
C   Array C contains integers cooresponding to the NOUT-1 steps
C

```

```

        DSAVG=(SL-DS1-DS2)/FLOAT(NOUT-3)
        A(1)=FLOAT(1)
        A(2)=FLOAT(NOUT)/2.0
        A(3)=FLOAT(NOUT-1)
        B(1)=DS1
        B(2)=DSAVG
        B(3)=DS2
        DO 20 J=1,NOUT-1
            C(J)=FLOAT(J)
20      CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        DSMIN=0.1*MIN(DS1,DS2)
C...CONVERGENCE LOOP FOR APPROPRIATE VALUE OF B(2)
C   The step size is splined with the slope held constant at the
C   ends. There must be NOUT-1 steps, with values of DS1 and
C   DS2 at the two ends. The mid-range step size is varied until
C   the sum of all the steps equals the arc length desired (SL).
C   By fixing the slope at the ends the program tries to ensure
C   that step sizes don't increase too rapidly.
        DO 100 KK=1,1000
            IERR=0
C...SPLINE SPACING WITH FIXED SLOPE AT THE ENDS
            CALL UGLYDK(3,0,0,A,B,0,0,CUB)
C...EVALUATE SPLINE TO FIND STEP SIZE AT INTERMEDIATE POINTS
            CALL EVALDK(3,NOUT-1,A,C,D,CUB)
C...CALCULATE LENGTH COVERED BY NEW SET OF STEPS
            SLC=0.0
            DO 40 I=1,NOUT-1
                SLC=SLC+D(I)
                IF (D(I).LT.0.0) IERR=1
40      CONTINUE
C...IF LENGTH IS NOT CLOSE TO TOTAL LENGTH NEEDED, SHIFT B(2)
C....AND REITERATE
            IF (IERR.EQ.1) THEN
                B(2)=0.0
            ELSE IF (ABS(SLC-SL).GT.DSMIN) THEN
                DIFF=SLC-SL
                B(2)=B(2)-DIFF/NOUT
            ELSE
                GOTO 101
            END IF
100     CONTINUE
            WRITE(*,*)'***WARNING*** FNSPLT DID NOT CONVERGE'
            WRITE(*,*)' ABS(SLC - SL) > TOL'
            WRITE(*,*)' SLC, SL, TOL = ',SLC,SL,DSMIN
            WRITE(*,*)NIN,NOUT,DS1,DS2
            IF (SLC.GT.SL) THEN
                WRITE(*,*)'Recommend using fewer points or smaller dS values'
            ELSE
                WRITE(*,*)'Recommend using more points of larger dS values'
            END IF
101     CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C...SHIFT LENGTH TO EXTEND FROM 0.0 TO 1.0
        E(1)=0.0
        DO 50 I=2,NOUT-1
            E(I)=D(I-1)/SLC+E(I-1)
50      CONTINUE
        E(NOUT)=1.0
C...EVALUATE OUTPUT POINTS AT CALCULATED INTERVAL
        CALL PEVALDK(NIN,NOUT,E,SI,XO,YO,CUB1,CUB2)
C
C
C#####

```

```

      RETURN
      END
C-----C
      SUBROUTINE DRIVDK(NIN,NOUT,XIN,XOUT,DYDX,D2YDX,A)
C  APRIL 1975 SPLINE PROGRAM SERIES  J.E.KERWIN
      REAL XIN(*),XOUT(*),DYDX(*),D2YDX(*),A(*)
      NM1=NIN-1
      J=1
      DO 3 N=1,NOUT
      IF(XOUT(N).GE.XIN(2)) GO TO 4
      J=1
      GO TO 5
4     IF(XOUT(N).LT.XIN(NM1)) GO TO 6
      J=NM1
      GO TO 5
6     IF(XOUT(N).GE.XIN(J+1)) GO TO 7
5     H1=XOUT(N)-XIN(J)
      H2=H1**2
      J2=J+NM1
      J3=J2+NM1
      DYDX(N)=3.0*A(J)*H2+2.0*A(J2)*H1+A(J3)
      D2YDX(N)=6.0*A(J)*H1+2.0*A(J2)
      GO TO 3
7     J=J+1
      GO TO 6
3     CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE EVALDK(NIN,NOUT,XIN,XOUT,YOUT,A)
C  APRIL 1975 SPLINE PROGRAM SERIES  J.E.KERWIN
      REAL XIN(*),XOUT(*),YOUT(*),A(*)
      NM1=NIN-1
      MOUT=IABS(NOUT)
      IF(NOUT.GT.0) GO TO 1
      DEL=(XIN(NIN)-XIN(1))/(MOUT-1)
      DO 2 N=1,MOUT
2     XOUT(N)=XIN(1)+(N-1)*DEL
1     J=1
      DO 3 N=1,MOUT
      IF(XOUT(N).GE.XIN(2)) GO TO 4
      J=1
      GO TO 5
4     IF(XOUT(N).LT.XIN(NM1)) GO TO 6
      J=NM1
      GO TO 5
6     IF(XOUT(N).GE.XIN(J+1)) GO TO 7
9     IF (XOUT(N).LT.XIN(J)) GO TO 8
5     H1=XOUT(N)-XIN(J)
      H2=H1**2
      H3=H1*H2
      J2=J+NM1
      J3=J2+NM1
      J4=J3+NM1
      YOUT(N)=A(J)*H3+A(J2)*H2+A(J3)*H1+A(J4)
      GO TO 3
7     J=J+1
      GO TO 6
8     J=J-1
      GO TO 9
3     CONTINUE
      RETURN
      END
C
C+++++

```

```

C      SUBROUTINE INTEDK(NIN,XIN,XL,XU,YDX,XYDX,XXYDX,A)
C      AUGUST 1975 SPLINE PROGRAM SERIES S.-K.TSAO
      REAL XIN(*),A(*)
      NM1=NIN-1
      IF(XL.LT.XIN(1)) GO TO 2
      DO 1 N=1,NIN
      IF(XL.GE.XIN(N)) GO TO 1
      JL=N-1
      GO TO 3
1      CONTINUE
      JL=NM1
      GO TO 3
2      JL=1
3      IF(XU.GE.XIN(NIN)) GO TO 4
      JU=1
      IF(XU.LE.XIN(1)) GO TO 6
      DO 5 N=JL,NIN
      IF(XU.GE.XIN(N)) GO TO 5
      JU=N-1
      GO TO 6
5      CONTINUE
      GO TO 6
4      JU=NM1
6      H1=XL-XIN(JL)
      H2=H1**2
      H3=H1*H2
      H4=H2**2
      H5=H2*H3
      H6=H3**2
      J1=JL
      J2=J1+NM1
      J3=J2+NM1
      J4=J3+NM1
      YDX=-A(J1)/4.0*H4-A(J2)/3.0*H3-A(J3)/2.0*H2-A(J4)*H1
      BUG=-A(J1)/5.0*H5-A(J2)/4.0*H4-A(J3)/3.0*H3-A(J4)/2.0*H2
      XYDX=BUG+XIN(J1)*YDX
      BUG=-A(J1)/6.0*H6-A(J2)/5.0*H5-A(J3)/4.0*H4-A(J4)/3.0*H3
      XXYDX=BUG+2.0*XIN(J1)*XYDX-XIN(J1)**2*YDX
      DO 7 N=JL,JU
      H1=XIN(N+1)-XIN(N)
      IF(N.EQ.JU) H1=XU-XIN(N)
      H2=H1**2
      H3=H1*H2
      H4=H2**2
      H5=H2*H3
      H6=H3**2
      J1=N
      J2=J1+NM1
      J3=J2+NM1
      J4=J3+NM1
      BUG=A(J1)/4.0*H4+A(J2)/3.0*H3+A(J3)/2.0*H2+A(J4)*H1
      YDX=YDX+BUG
      CAT=A(J1)/5.0*H5+A(J2)/4.0*H4+A(J3)/3.0*H3+A(J4)/2.0*H2
      PIG=CAT+XIN(J1)*BUG
      XYDX=XYDX+PIG
      DOG=A(J1)/6.0*H6+A(J2)/5.0*H5+A(J3)/4.0*H4+A(J4)/3.0*H3
      XXYDX=XXYDX+DOG+2.0*XIN(J1)*PIG-XIN(J1)**2*BUG
7      CONTINUE
      RETURN
      END
C
C+++++
C      SUBROUTINE LINDK(NIN,XIN,YIN,A)

```

```

C-----GENERATES CUBIC COEFFICIENTS FOR PIECEWISE LINEAR FIT-----
      REAL XIN(*),YIN(*),A(*)
      NM1=NIN-1
      DO 1 N=1,NM1
        A(N)=0.0
        M=N+NM1
        A(M)=0.0
        M=M+NM1
        A(M)=(YIN(N+1)-YIN(N))/(XIN(N+1)-XIN(N))
        M=M+NM1
1      A(M)=YIN(N)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE PEVALDK(NIN,NOUT,ARK,S,XO,YO,CUB1,CUB2)
      REAL XO(200),YO(200),S(200),ARK(200)
      REAL CUB1((200-1)*4),CUB2((200-1)*4)
      CALL EVALDK(NIN,NOUT,S,ARK,XO,CUB1)
      CALL EVALDK(NIN,NOUT,S,ARK,YO,CUB2)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE PUGLYDK(NIN,X,Y,S,CUB1,CUB2)
      REAL X(200),Y(200),S(200)
      REAL CUB1((200-1)*4),CUB2((200-1)*4)
      STOT=0.0
      S(1)=0.0
      DO 10 I=2,NIN
        S(I)=SQRT((X(I)-X(I-1))**2+(Y(I)-Y(I-1))**2)+STOT
        STOT=STOT+SQRT((X(I)-X(I-1))**2+(Y(I)-Y(I-1))**2)
10     CONTINUE
      DO 20 I=2,NIN
        S(I)=S(I)/STOT
20     CONTINUE
      NCL=1
      NCR=1
      ESL=0
      ESR=0
      CALL UGLYDK(NIN,NCL,NCR,S,X,ESL,ESR,CUB1)
      CALL UGLYDK(NIN,NCL,NCR,S,Y,ESL,ESR,CUB2)
      RETURN
      END
C-----
      SUBROUTINE UGLYDK(NIN,NCL,NCR,XIN,YIN,ESL,ESR,AE)
C-----1975 DUCK SERIES J.E.KERWIN MODIFIED 6/21/82-----
C-----TRI-DIAGONAL MATRIX SOLUTION BUILT IN-----
      REAL XIN(*),YIN(*),AE(*),H(200),D(200),AU(200),AM(200),
      *      S(200),AL(200),X(200)
      DATA HALF/0.5E00/,TWO/2.0E00/,SIX/6.0E00/,RAD/1.745329E-02/
      NM1=NIN-1
      NM2=NM1-1
      NM3=NM2-1
      NEQ=NM2
      DO 1 N=1,NM1
        H(N)=XIN(N+1)-XIN(N)
1      D(N)=(YIN(N+1)-YIN(N))/H(N)
        IF(NCL.EQ.2) NEQ=NEQ+1
        IF(NCR.EQ.2) NEQ=NEQ+1
        NSQ=NEQ**2
        J=1
        IF(NCL.LT.2) GO TO 6
        AM(1)=TWO*H(1)
        AU(1)=H(1)
        SLP=ESL*RAD
        S(1)=(D(1)-TAN(SLP))*SIX

```



```

      J=J+1
      AL(2)=H(1)
6    DO 5 N=1,NM2
      IF(N.GT.1) AU(J-1)=H(N)
      AM(J)=TWO*(H(N)+H(N+1))
      IF(N.LT.NM2) AL(J+1)=H(N+1)
      IF(N.EQ.2.AND.NCL.EQ.1) AU(J-1)=AU(J-1)-H(N-1)**2/H(N)
      IF(N.EQ.1.AND.NCL.EQ.1) AM(J)=AM(J)+(1.0+H(N)/H(N+1))*H(N)
      IF(N.EQ.NM2.AND.NCR.EQ.1) AM(J)=AM(J)+(1.0+H(N+1)/H(N))*H(N+1)
      IF(N.EQ.NM3.AND.NCR.EQ.1) AL(J+1)=AL(J+1)-H(N+2)**2/H(N+1)
      S(J)=(D(N+1)-D(N))*SIX
      J=J+1
5    CONTINUE
      IF(NCR.LT.2) GO TO 7
      AL(NEQ)=-H(NM1)
      AM(NEQ)=-TWO*H(NM1)
      AU(NEQ-1)=H(NM1)
      SLP=ESR*RAD
      S(J)=(D(NM1)+TAN(SLP))*SIX
7    CONTINUE
      DO 4 K=2,NEQ
      AL(K)=AL(K)/AM(K-1)
      AM(K)=AM(K)-AL(K)*AU(K-1)
      S(K)=S(K)-AL(K)*S(K-1)
4    CONTINUE
      X(NEQ)=S(NEQ)/AM(NEQ)
      DO 2 L=2,NEQ
      K=NEQ-L+1
      X(K)=(S(K)-AU(K)*X(K+1))/AM(K)
2    CONTINUE
      DO 22 N=1,NEQ
22   S(N)=X(N)
      HOLD=S(NEQ)
      IF(NCL.EQ.2) GO TO 8
      DO 9 N=1,NM2
      M=NM2-N+2
9    S(M)=S(M-1)
      IF(NCL.EQ.0) S(1)=0.0
      BUG=H(1)/H(2)
      IF(NCL.EQ.1) S(1)=(1.0+BUG)*S(2)-BUG*S(3)
8    IF(NCR.EQ.0) S(NIN)=0.0
      BUG=H(NM1)/H(NM2)
      IF(NCR.EQ.1) S(NIN)=(1.0+BUG)*S(NM1)-BUG*S(NM2)
      IF(NCR.EQ.2) S(NIN)=HOLD
      DO 10 N=1,NM1
      AE(N)=(S(N+1)-S(N))/(SIX*H(N))
      M=N+NM1
      AE(M)=HALF*S(N)
      M=M+NM1
      AE(M)=D(N)-H(N)*(TWO*S(N)+S(N+1))/SIX
      M=M+NM1
10   AE(M)=YIN(N)
      RETURN
      END

```

Appendix B

Sample FIT2D Input Files

B.1 Bounded Foil (*fname.ctrl*)

```
Header: Sample for Cupped B-1
Foil geometry file name
rcup.foil
AOA Xpiv Ypiv
0.5 0.3 0.01
Chord/Tunnel Width
0.9
USL DSL PHI1 PHI2
2.0 2.0 0.5 0.25
NUS NDS NVS NTOP NBOT(Always 8*n-1 points)
87 87 95 95 95
RESLE RESMID RESTE RESWAL RESBL PACK
0.001 0.03 0.001 1.0e-5 1.0e-5 0.20
Re#(chord based)
3e6
Desired Inmesh Input file:
cupi.dat
Desired Inmesh Restart file
cupr.dat
Number of INMESH Iterations:
1500
Convergence tolerance:
1.0e-10
Wake Data(-1 none, 0 VLM, or nranswk)
10
0.713324 -0.016042
0.772837 -0.022880
0.879395 -0.032600
1.027451 -0.043110
1.210484 -0.052885
1.424693 -0.061495
1.674272 -0.069320
1.953325 -0.077086
2.257376 -0.083998
2.580143 -0.089997
```

B.2 Unbounded Foil (*fname.ctrl*)

Header: Sample Unbounded Cupped B-1 Foil
Foil geometry filename:
rcup.foil
ADA Xpiv Ypiv
0.5 0.3 0.01
Chord/Tunnel Width
0.08
USL DSL PHI1 PHI2
5.0 5.0 2.00 1.00
NUS NDS NVS NTOP NBOT(Always 8*n-1 points)
87 87 95 95 95
RESLE RESMID RESTE RESWAL RESBL PACK
0.001 0.03 0.001 0.005 1.0e-5 0.20
Re#(chord based)
3e6
Desired Inmesh Input file:
cupi.dat
Desired Inmesh Restart file
cupr.dat
Number of INMESH Iterations:
1500
Convergence tolerance:
1.0e-10
Wake Data(-1 none, 0 VLM, or nranswk)
20
0.713324 -0.016042
0.772837 -0.022880
0.879395 -0.032600
1.027451 -0.043110
1.210484 -0.052885
1.424693 -0.061495
1.674272 -0.069320
1.953325 -0.077086
2.257376 -0.083998
2.580143 -0.089997
2.917188 -0.095028
3.261140 -0.099227
3.603834 -0.102689
3.939133 -0.105482
4.261238 -0.107661
4.565246 -0.109321
4.846685 -0.110522
5.101448 -0.111331
5.326634 -0.111836
5.518953 -0.112118

B.3 Sample Foil Geometry File (*fname.foil*)

Offsets start at the trailing edge marching forward on lowersurface, around the leading edge and along upper surface to trailing edge. It is not necessary to have an explicit point at the leading edge. FIT2D splines the offsets and finds the leading edge. FIT2D assumes that this input file is at zero degrees angle of attack.

Cupped B-1 Foil

2

1.0

161	
1.000000	0.000000
0.989517	0.002052
0.979110	0.003558
0.968851	0.004586
0.958810	0.005204
0.949059	0.005478
0.939668	0.005477
0.930708	0.005269
0.922250	0.004920
0.914365	0.004499
.	.
.	.
.	.
0.007665	-0.006030
0.005500	-0.005229
0.003652	-0.004320
0.002127	-0.003289
0.000963	-0.002132
0.000220	-0.000846
0.000041	0.000570
0.000133	0.001781
0.000676	0.003074
0.001617	0.004450
0.002986	0.005909
0.004814	0.007453
0.007129	0.009082
0.009962	0.010796
.	.
.	.
.	.
0.953672	0.028109
0.962481	0.024014
0.971452	0.019244
0.980651	0.013711
0.990145	0.007326
1.000000	0.000000

Appendix C

Marine Hydrodynamics Lab Water Tunnel Geometry

C.1 System Overview

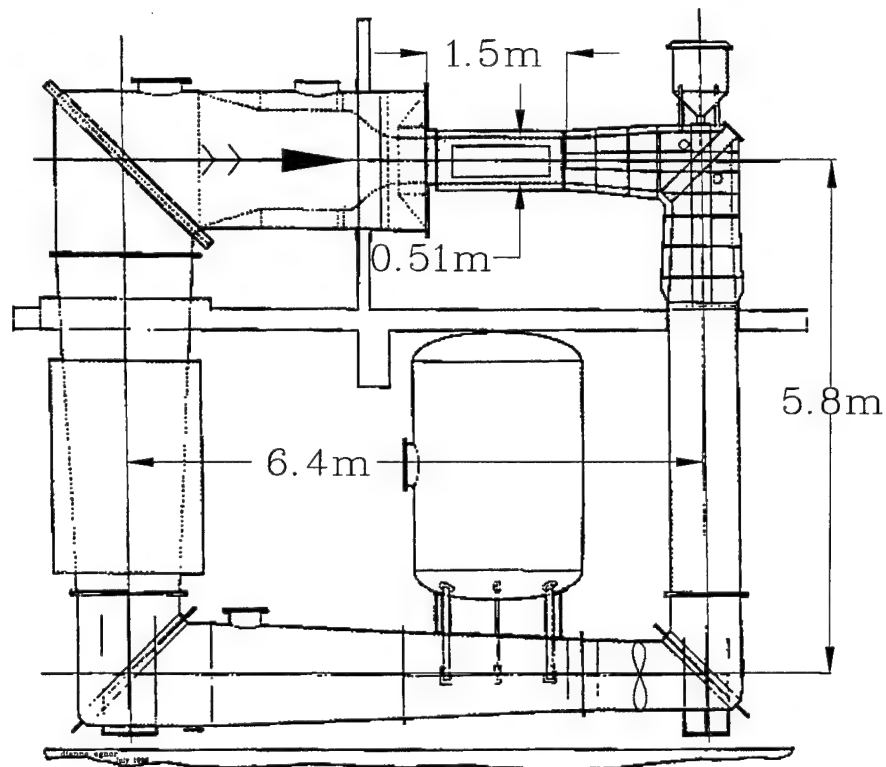


Figure C-1: MIT MHL Water Tunnel

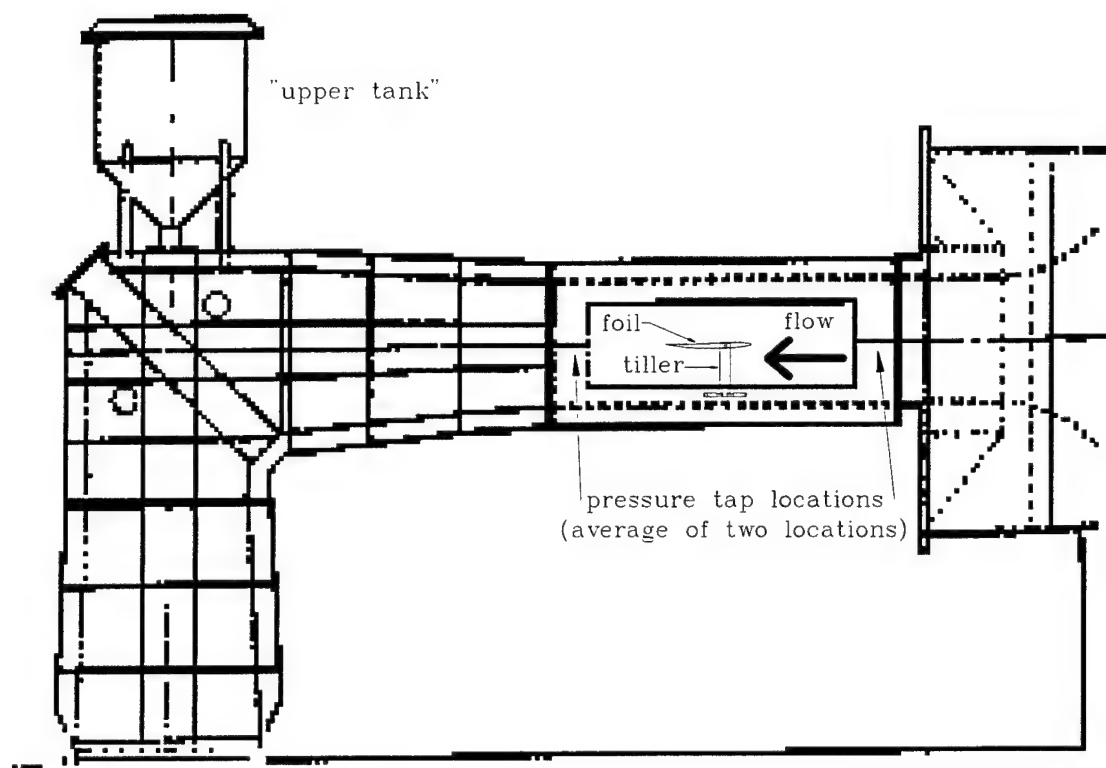


Figure C-2: MIT MHL Water Tunnel Test Section

Appendix D

GETWAKE Program Listing

```
PROGRAM GETWAKE
C
C
C Reads in a set of wakepoints .
C
C INPUT REQUIREMENTS:
C JUNK header line
C NPOINTS - NUMBER OF OFFSETS
C XIN, YIN
C .
C (to end of file)
C
C
C
C
C IMPLICIT NONE
C REAL XIN(1000),YIN(1000)
C REAL Xout(200),Yout(200)
C integer npoints,nfoil,i,k,ntemp,nout,j,ierr
C character FNOOPEN*20
C character junk*30, title*30,JUNK2*2
C character PROMPT2*30
C character FIN*20
C WRITE(*,*)'***'
C write(*,'(A,$)') 'Enter Filename of Tecplot stream data: '
C read(*,'(A)') FNOOPEN
C WRITE(*,*)'***'
C write(*,'(A)') 'Shortening file:' // FNOOPEN
C OPEN(UNIT=1,FILE=FNOOPEN,FORM='FORMATTED',STATUS='OLD')
C WRITE(*,*)'***'
C WRITE(*,*)' ***READING DATA IN***'
C WRITE(*,*)'***'
C read(1,'(A)') junk
C read(1,'(A)') junk
C read(1,'(A)') junk
C WRITE(*,*)'READ THE TOP OF THE FILE'
C
C
C keep next read as the title
C read(1,'(A)') title
C
C
C now want to extract the integer for number
C of pairs of points
C CALL READIJ(1,npoints,J,IERR)
98 FORMAT(A23,I6,A5)
C WRITE(*,98)'***NUMBER DATAPOINTS IN: ',NPOINTS,' ***'
C WRITE(*,*)'***'
```

```

      READ(1,'(A)') JUNK
      READ(1,*) ((xin(k),yin(k)),K=1,npoints)
C      READ(1,*) JUNK
      CLOSE(1)
      WRITE(*,'(A)') ' ***CLOSING FILE: ' // FOPEN // ' ***'
      WRITE(*,*)'***'

C
C      Shorten Data file by taking only 20 datapoints in range
C
      NOUT=20
      WRITE(*,98)'***NUMBER OF POINTS OUT: ',NOUT,' ***'
      WRITE(*,*)'***'

      NTEMP=int(float(npoints)/NOUT)
      WRITE(*,98)'***DATA SKIP INTERVAL: ',NTEMP,' ***'
      WRITE(*,*)'***'

      do 100 i=1,NOUT
        xout(i)=xin(i*ntemp-ntemp+1)
        yout(i)=yin(i*ntemp-ntemp+1)
100      continue

      write(*,*)'****Data has been shortened.*****'
      write(*,*)'***'

      PROMPT2 = 'Output FILE'// ' (//wakelin.dat//)' = '
      WRITE (*,'(A,$)') PROMPT2
      READ (*,'(A)') FIN
      IF (FIN(1:1).EQ.' ') FIN = 'wakelin.dat'
      WRITE (*,'(A)') 'OPENING FILE: ' // FIN
      OPEN (UNIT=3,FILE=FIN,STATUS='UNKNOWN')

91      format(i4)
92      format(2f10.6)
      WRITE(*,*)'***'
      WRITE(*,*)'***WRITING OUTPUT TO FILE***'
      WRITE(*,*)'***'
      WRITE(3,*)'Shortened: ',TITLE
      WRITE(3,91)NOUT
      do 200 i=1,nout
        write(3,92) xout(i),yout(i)
200      continue
      close(3)
      write(*,*)' All Done, Thanks. '
      WRITE(*,*)'***'
      stop
      end

cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C
C
***** SUBROUTINE READIJ *****
*   EXTRACTS THE INTEGER I,J INDICES FROM A TECPLOT FILE HEADER *
*   Arguments:  IUNIT....Unit number of Tecplot file           *
*               I.....Number of columns in IJ ordered data    *
*               J.....Number of rows in IJ ordered data       *
*               IERR....0=Valid data, 1=Subroutine failed      *
*   It is assumed that I and J are between 1 and 9999          *
*               Justin E. Kerwin   August 19,1993              *
*****
SUBROUTINE READIJ(IUNIT,I,J,IERR)
CHARACTER*80 LABEL
CHARACTER*4 ICODE
ICODE=' '

```



```

      IERR=0
C.....Read in the Tecplot header line as a character string LABEL.
      READ(IUNIT,'(A)') LABEL
C.....Find the length of the character string.....
      LMAX=LEN(LABEL)
      LBEGIN=1
C.....First find I:(IJ=1), then find J:(IJ=2).....
      DO 400 IJ=1,2
        ICODE=' '
C.....Find the position of the first = sign in the string.....
        DO 100 L=LBEGIN,LMAX
          IF(LABEL(L:L).EQ.CHAR(61)) THEN
            LMIN=L+1
            GO TO 110
          END IF
100      CONTINUE
          IERR=1
          GO TO 9999
110      CONTINUE
C.....Find the position of the first number following an = sign.
        DO 200 L=LMIN,LMAX
          IF(LABEL(L:L).GT.CHAR(48).AND.LABEL(L:L).LT.CHAR(58)) THEN
            LSTART=L
            GO TO 210
          END IF
200      CONTINUE
          IERR=1
          GO TO 9999
210      K=1
C.....Generate a substring ICODE consisting of consecutive numbers
        DO 300 L=LSTART,LSTART+4
          IF(LABEL(L:L).LT.CHAR(48).OR.LABEL(L:L).GT.CHAR(57)) THEN
            LBEGIN=L
            GO TO 310
          ELSE
            ICODE(K:K)=LABEL(L:L)
            K=K+1
          END IF
300      CONTINUE
          IERR=1
          GO TO 9999
C.....Convert the substring to integers I,J using an internal read
310      IF(IJ.EQ.1) READ(ICODE,'(I4)') I
          IF(IJ.EQ.2) READ(ICODE,'(I4)') J
400      CONTINUE

9999 RETURN
      END

```

Appendix E

PATCH Program Listing

```
PROGRAM PATCH
REAL*8 X(200,200),Y(200,200)
character*10 FNOOPEN
  FNOOPEN='inp02.dat'
  OPEN(UNIT=1,FILE=FNOOPEN,FORM='FORMATTED',STATUS='OLD')
  READ(1,*) JGRD,KGRD
  READ(1,*) ((X(J,K),J=1,JGRD),K=1,KGRD)
  READ(1,*) ((Y(J,K),J=1,JGRD),K=1,KGRD)
  CLOSE(1)

  FNOOPEN='inp02.dat'
  OPEN(UNIT=2,FILE=FNOOPEN,FORM='FORMATTED',STATUS='UNKNOWN')
  WRITE(2,*) JGRD-1,KGRD
  WRITE(2,*) ((X(J,K),J=2,JGRD),K=1,KGRD)
  WRITE(2,*) ((Y(J,K),J=2,JGRD),K=1,KGRD)
  CLOSE(2)

  FNOOPEN='inp03.dat'
  OPEN(UNIT=1,FILE=FNOOPEN,FORM='FORMATTED',STATUS='OLD')
  READ(1,*) JGRD,KGRD
  READ(1,*) ((X(J,K),J=1,JGRD),K=1,KGRD)
  READ(1,*) ((Y(J,K),J=1,JGRD),K=1,KGRD)
  CLOSE(1)

  FNOOPEN='inp03.dat'
  OPEN(UNIT=2,FILE=FNOOPEN,FORM='FORMATTED',STATUS='UNKNOWN')
  WRITE(2,*) JGRD-1,KGRD
  WRITE(2,*) ((X(J,K),J=2,JGRD),K=1,KGRD)
  WRITE(2,*) ((Y(J,K),J=2,JGRD),K=1,KGRD)
  CLOSE(2)

  FNOOPEN='inp05.dat'
  OPEN(UNIT=1,FILE=FNOOPEN,FORM='FORMATTED',STATUS='OLD')
  READ(1,*) JGRD,KGRD
  READ(1,*) ((X(J,K),J=1,JGRD),K=1,KGRD)
  READ(1,*) ((Y(J,K),J=1,JGRD),K=1,KGRD)
  CLOSE(1)

  FNOOPEN='inp05.dat'
  OPEN(UNIT=2,FILE=FNOOPEN,FORM='FORMATTED',STATUS='UNKNOWN')
  WRITE(2,*) JGRD-1,KGRD
  WRITE(2,*) ((X(J,K),J=2,JGRD),K=1,KGRD)
  WRITE(2,*) ((Y(J,K),J=2,JGRD),K=1,KGRD)
  CLOSE(2)
```

```

FNOOPEN='inp06.dat'
OPEN(UNIT=1,FILE=FNOOPEN,FORM='FORMATTED',STATUS='OLD')
READ(1,*) JGRD,KGRD
READ(1,*) ((X(J,K),J=1,JGRD),K=1,KGRD)
READ(1,*) ((Y(J,K),J=1,JGRD),K=1,KGRD)
CLOSE(1)

FNOOPEN='inp06.dat'
OPEN(UNIT=2,FILE=FNOOPEN,FORM='FORMATTED',STATUS='UNKNOWN')
WRITE(2,*) JGRD-1,KGRD
WRITE(2,*) ((X(J,K),J=2,JGRD),K=1,KGRD)
WRITE(2,*) ((Y(J,K),J=2,JGRD),K=1,KGRD)
CLOSE(2)

stop
end

```

Appendix F

Case Study Foil Offsets

Offsets start at the trailing edge of the pressure side and march forward around the leading edge to the trailing edge of the suction side. Both data sets use a normalized chord length 1.0. The offsets listed here correspond to the foils displayed in Figures 5-1 and 5-9.

F.1 Case Study I: The HRA Foil

1.0	0.0	0.550	0.047207
0.995	-0.000896	0.600	0.045990
0.990	-0.000797	0.650	0.044037
0.985	-0.000703	0.700	0.041199
0.975	-0.000537	0.725	0.039327
0.965	-0.000397	0.750	0.037072
0.950	-0.000225	0.775	0.034427
0.925	-0.000031	0.800	0.031406
0.900	0.000049	0.825	0.028036
0.875	0.000009	0.850	0.024371
0.850	-0.000150	0.875	0.020515
0.825	-0.000405	0.900	0.016571
0.800	-0.000734	0.925	0.012588
0.775	-0.001126	0.950	0.008599
0.750	-0.001574	0.965	0.006225
0.725	-0.002076	0.975	0.004671
0.700	-0.002631	0.985	0.003166
0.650	-0.003968	0.990	0.002433
0.600	-0.005685	0.995	0.001713
0.550	-0.007780	1.000	0.0
0.500	-0.010205		
0.450	-0.012882		
0.400	-0.015697		
0.350	-0.018500		
0.300	-0.021093		
0.250	-0.023204		
0.200	-0.024436		
0.175	-0.024512		
0.150	-0.024077		
0.125	-0.023054		
0.100	-0.021373		
0.075	-0.018969		
0.050	-0.015681		
0.035	-0.013111		
0.025	-0.011017		
0.015	-0.008420		
0.010	-0.006793		
0.005	-0.004709		
0.0025	-0.003274		
0.001	-0.002036		
0.0	0.0		
0.001	0.002076		
0.0025	0.003375		
0.005	0.004911		
0.010	0.007200		
0.015	0.009036		
0.025	0.012061		
0.035	0.014599		
0.050	0.017865		
0.075	0.022406		
0.100	0.026204		
0.125	0.029474		
0.150	0.032329		
0.175	0.034841		
0.200	0.037059		
0.250	0.040736		
0.300	0.043546		
0.350	0.045599		
0.400	0.046962		
0.450	0.047677		
0.500	0.047760		

F.2 B-1 Foil With Cup Modification To Trailing Edge

1.000000	0.000000	0.069396	-0.012064
0.989517	0.002052	0.060654	-0.011696
0.979110	0.003558	0.053133	-0.011315
0.968851	0.004586	0.046637	-0.010923
0.958810	0.005204	0.040974	-0.010524
0.949059	0.005478	0.035949	-0.010120
0.939668	0.005477	0.031370	-0.009713
0.930708	0.005269	0.027095	-0.009301
0.922250	0.004920	0.023110	-0.008873
0.914365	0.004499	0.019418	-0.008416
0.907121	0.004072	0.016024	-0.007916
0.900497	0.003672	0.012931	-0.007361
0.894374	0.003298	0.010143	-0.006736
0.888628	0.002943	0.007665	-0.006030
0.883131	0.002604	0.005500	-0.005229
0.877761	0.002274	0.003652	-0.004320
0.872391	0.001950	0.002127	-0.003289
0.866896	0.001626	0.000963	-0.002132
0.861152	0.001297	0.000220	-0.000846
0.855033	0.000959	0.000000	0.000570
0.848423	0.000606	0.000133	0.001781
0.841250	0.000236	0.000676	0.003074
0.833456	-0.000154	0.001617	0.004450
0.824986	-0.000568	0.002986	0.005909
0.815783	-0.001006	0.004814	0.007453
0.805789	-0.001473	0.007129	0.009082
0.794947	-0.001971	0.009962	0.010796
0.783201	-0.002503	0.013344	0.012596
0.770495	-0.003071	0.017305	0.014480
0.756770	-0.003678	0.021881	0.016443
0.742000	-0.004322	0.027105	0.018482
0.726196	-0.004997	0.033011	0.020591
0.709371	-0.005694	0.039635	0.022766
0.691542	-0.006406	0.047010	0.025003
0.672721	-0.007126	0.055170	0.027297
0.652924	-0.007845	0.064149	0.029643
0.632166	-0.008556	0.073983	0.032038
0.610461	-0.009251	0.084704	0.034476
0.587823	-0.009922	0.096348	0.036953
0.564277	-0.010563	0.108948	0.039464
0.539906	-0.011168	0.122492	0.042000
0.514822	-0.011734	0.136919	0.044547
0.489138	-0.012255	0.152161	0.047087
0.462964	-0.012730	0.168152	0.049605
0.436413	-0.013153	0.184827	0.052085
0.409596	-0.013521	0.202117	0.054513
0.382625	-0.013830	0.219958	0.056871
0.355611	-0.014076	0.238283	0.059145
0.328667	-0.014255	0.257024	0.061319
0.301935	-0.014366	0.276117	0.063376
0.275618	-0.014414	0.295494	0.065302
0.249922	-0.014402	0.315089	0.067080
0.225056	-0.014337	0.334845	0.068702
0.201226	-0.014221	0.354719	0.070169
0.178641	-0.014061	0.374667	0.071483
0.157509	-0.013861	0.394646	0.072647
0.138036	-0.013626	0.414616	0.073662
0.120430	-0.013359	0.434532	0.074531
0.104875	-0.013067	0.454353	0.075256
0.091313	-0.012752	0.474035	0.075838
0.079551	-0.012417	0.493537	0.076282
		0.512816	0.076587
		0.531828	0.076757
		0.550533	0.076795
		0.568899	0.076701

0.586921	0.076477
0.604596	0.076125
0.621920	0.075647
0.638890	0.075044
0.655504	0.074317
0.671758	0.073469
0.687648	0.072500
0.703172	0.071413
0.718327	0.070208
0.733108	0.068888
0.747514	0.067454
0.761542	0.065910
0.775192	0.064268
0.788463	0.062543
0.801358	0.060746
0.813876	0.058891
0.826017	0.056991
0.837782	0.055059
0.849171	0.053108
0.860186	0.051152
0.870825	0.049203
0.881089	0.047275
0.890979	0.045380
0.900510	0.043511
0.909739	0.041590
0.918732	0.039530
0.927555	0.037241
0.936276	0.034634
0.944959	0.031619
0.953672	0.028109
0.962481	0.024014
0.971452	0.019244
0.980651	0.013711
0.990145	0.007326
1.000000	0.000000

Bibliography

- [1] I. H. Abbott and A. E. Von Doenhoff. *Theory of Wing Sections*. Dover, New York, 1959.
- [2] D. A. Anderson, J. C. Tannehill, and R. H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Hemisphere Publishing, 1984.
- [3] J. W. Bailar, S. D. Jessup, and Y. T. Shen. Improvement of Surface Ship Propeller Cavitation Performance Using Advanced Blade Sections. In *23rd American Towing Tank Conference*, New Orleans, USA, 1992.
- [4] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 2nd edition, 1996.
- [5] S. D. Black. FNSPLT Cubic Point Distribution Program. Unpublished, 1996.
- [6] S.D. Black. An Integrated Lifting Surface/Navier-Stokes Propulsor Design Method. Master's thesis, MIT Department of Ocean Engineering, June 1994.
- [7] F. Bloch. Effect of Trailing Edge Geometry on Propulsive Performance. Technical report, Dept. of Ocean Engineering, Massachusetts Institute of Technology, December 1994.
- [8] R. M. Coleman. INMESH: An Interactive Program for Numerical Grid Generation. Technical Report DTNSRDC-85/054, David W. Taylor Naval Ship Research and Development Center, August 1985.

- [9] W. B. Coney. Some Notes on the Calculation of Viscous Effects on Lift. Technical Report 89-8, Department of Ocean Engineering, Massachusetts Institute of Technology, September 1989.
- [10] M. Drela. XFOIL: An Analysis and Design System for Low Reynolds Number Aerofoils. *Lecture Notes in Engineering: Low Reynolds Number Aerodynamics*, (54), 1989.
- [11] M. Drela. XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils. In *Proceedings of the Conference on Low Reynolds Number Aerodynamics*, 1989.
- [12] V. M. Falkner. The Solution of Lifting Plane Problems by Vortex Lattice Theory. R & M 2591, Aeronautical Research Council, 1947.
- [13] J.J. Gorski. Incompressible Cascade Calculation using an Upwind Differenced TVD Scheme. Presented to ASME Winter Annual Meeting, November 1988.
- [14] F. B. Hildebrand. *Advanced Calculus for Applications*. Prentice-Hall, Inc, second edition, 1976.
- [15] G. S. Hufford, M. Drela, and J. E. Kerwin. Viscous Flow around Marine Propellers using Boundary-Layer Strip Theory. *Journal of Ship Research*, 38(1), March 1994.
- [16] J. Jorde. A Study of Two Cambered Trailing Edge Geometries. Technical report, Dept. of Ocean Engineering, Massachusetts Institute of Technology, December 1995.
- [17] J. E. Kerwin. The MIT Marine Hydrodynamics Water Tunnel—A 53rd Anniversary Celebration. Technical report, New England Section, SNAME, May 1992.
- [18] Justin E. Kerwin. 13.04 Lecture Notes - Hydrofoils and Propellers, February 1994.

- [19] R. W. Kimball and D. E. Egnor. HRA 2D Foil Experiment. Technical Report 97-2, Department of Ocean Engineering, Massachusetts Institute of Technology, 1997. in preparation.
- [20] Spyros A. Kinnas. Hydrofoil Lift and Drag From Momentum Integrations. Technical Report Rep. 91-4, Department of Ocean Engineering, Massachusetts Institute of Technology, November 1991.
- [21] J. T. Lee. A Potential Based Panel Method for the Analysis of Marine Propellers in Steady Flow. Technical Report 87-13, Dept. of Ocean Engineering, Massachusetts Institute of Technology, July 1987.
- [22] J. T. Lee. *A Potential Based Panel Method for The Analysis of Marine Propellers in Steady Flow*. PhD thesis, Department of Ocean Engineering, Massachusetts Institute of Technology, August 1987.
- [23] E. V. Lewis. *Principles of Naval Architecture*. SNAME, Jersey City, NJ, 1988.
- [24] J. N. Newman. *Marine Hydrodynamics*. The MIT Press, Cambridge, Massachusetts, 1977.
- [25] P. Nguyen and J. Gorski. Navier-Stokes Analysis of Turbulent Boundary Layer and Wake for Two-Dimensional Lifting Bodies. In *Proceedings of the Eighteenth Symposium on Naval Hydrodynamics*, pages 633-644. Office of Naval Research, January 1991.
- [26] P.N. Nguyen. Use of Navier-Stokes Analysis in Section Design. Technical Report DTRC-SHD-1262-04, David Taylor Research Center, December 1990. (UNCLASSIFIED).
- [27] Phillip Colella & Elbridge Gerry Puckett. *Modern Numerical Methods for Fluid Flow*, November 1994.

- [28] R. H. Sabersky, A. J. Acosta, and E. G. Hauptman. *Fluid Flow, A first Course In Fluid Mechanics*. Macmillan Publishing Company, 3rd edition, 1989.
- [29] D. Valentine. Reynolds-Averaged Navier-Stokes Codes and Marine Propulsor Analysis. Technical Report Report HD-1262-06, Carderock Division, Naval Surface Warfare Center, October 1993.
- [30] F. M. White. *Viscous Fluid Flow*. McGraw-Hill, second edition, 1991.
- [31] D. K. P. Yue. 13.021 Lecture Notes - Marine Hydrodynamics I, September 1994.